intel

Microsystem Components Handbook

Peripherals Volume II

Order Number: 230843-003

intel®

LITERATURE

To order Intel Literature write or call:

Intel Literature Sales
P.O. Box 58130
Santa Clara, CA 95052-8130

Intel Literature Sales:
(800) 548-4725
Other Inquiries:
(800) 538-1876

Use the order blank on the facing page or call our Toll Free Number listed above to order literature. Remember to add your local sales tax and a 10% handling charge for U.S. customers, 20% for Canadian customers.

1986 HANDBOOKS

Product Line handbooks contain data sheets, application notes, article reprints and other design information.

NAME	ORDER NUMBER	*PRICE IN U.S. DOLLARS
COMPLETE SET OF 9 HANDBOOKS Get a 30% discount off the retail price of $171.00	231003	**$120.00**
MEMORY COMPONENTS HANDBOOK	210830	**$18.00**
MICROCOMMUNICATIONS HANDBOOK	231658	**$18.00**
MICROCONTROLLER HANDBOOK	210918	**$18.00**
MICROSYSTEM COMPONENTS HANDBOOK Microprocessor and peripherals (2 Volume Set)	230843	**$25.00**
DEVELOPMENT SYSTEMS HANDBOOK	210940	**$18.00**
OEM SYSTEMS HANDBOOK	210941	**$18.00**
SOFTWARE HANDBOOK	230786	**$18.00**
MILITARY HANDBOOK	210461	**$18.00**
QUALITY/RELIABILITY HANDBOOK	210997	**$20.00**
PRODUCT GUIDE Overview of Intel's complete product lines	210846	**No charge**
LITERATURE GUIDE Listing of Intel Literature	210620	**No charge**
INTEL PACKAGING SPECIFICATIONS Listing of Packaging types, number of leads, and dimensions	231369	**No charge**

*These prices are for the U. S. and Canada only. In Europe and other international locations, please contact your local Intel Sales Office or Distributor for literature prices.

intel®

U.S. LITERATURE ORDER FORM

NAME: _____

COMPANY: _____

ADDRESS: _____

CITY: _____ STATE: _____ ZIP: _____

COUNTRY: _____

PHONE NO.: ( ) _____

ORDER NO.	TITLE	QTY.	PRICE	TOTAL
☐☐☐☐☐☐ - ☐☐☐	_____	____ ×	____ =	_____
☐☐☐☐☐☐ - ☐☐☐	_____	____ ×	____ =	_____
☐☐☐☐☐☐ - ☐☐☐	_____	____ ×	____ =	_____
☐☐☐☐☐☐ - ☐☐☐	_____	____ ×	____ =	_____
☐☐☐☐☐☐ - ☐☐☐	_____	____ ×	____ =	_____
☐☐☐☐☐☐ - ☐☐☐	_____	____ ×	____ =	_____
☐☐☐☐☐☐ - ☐☐☐	_____	____ ×	____ =	_____

Add appropriate postage and handling to subtotal
10% U.S.
20% Canada

Subtotal _____

Your Local Sales Tax _____

Postage & Handling _____

Total _____

Allow 2-4 weeks for delivery

Pay by Visa, MasterCard, Check or Money Order, payable to Intel Books. Purchase Orders have a $50.00 minimum

☐ Visa ☐ MasterCard Expiration Date _____

Account No. _____

Signature: _____

Mail To: Intel Literature Sales
P.O. Box 58130
Santa Clara, CA
95052-8130

Customers outside the U.S. and Canada should contact the local Intel Sales Office or Distributor listed in the back of most Intel literature.

Call Toll Free: (800) 548-4725 for phone orders

Prices good until 12/31/86.

Source HB

Mail To: Intel Literature Sales
P.O. Box 58130
Santa Clara, CA 95052-8130

MICROSYSTEM COMPONENTS HANDBOOK

1986

About Our Cover:
The design on our front cover is an abstract portrayal of the unlimited interface linking options available with Intel microsystem components. Intel microprocessors and associated peripherals are the building blocks which provide total systems development solutions. Intel's superior technology, reliability and support provides easier solutions to specific development problems. Thereby, cutting "time-to-market" and creating a greater market share.

Intel Corporation makes no warranty for the use of its products and assumes no responsibility for any errors which may appear in this document nor does it make a commitment to update the information contained herein.

Intel retains the right to make changes to these specifications at any time, without notice.

Contact your local sales office to obtain the latest specifications before placing your order.

The following are trademarks of Intel Corporation and may only be used to identify Intel Products:

Above, BITBUS, COMMputer, CREDIT, Data Pipeline, GENIUS, i, $\hat{i}$, ICE, iCEL, iCS, iDBP, iDIS, I²ICE, iLBX, i_m, iMDDX,, iMMX, Insite, Intel, int_el, int_elBOS, Intelevision, $int_eligent$ Identifier, $int_eligent$ Programming, Intellec, Intellink, iOSP, iPDS, iPSC, iRMX, iSBC, iSBX, iSDM, iSXM, KEPROM, Library Manager, MCS, Megachassis, MICROMAINFRAME, MULTIBUS, MULTICHANNEL, MULTIMODULE, ONCE, OpenNET, Plug-A-Bubble, PROMPT, Promware, QUEST, QueX, Ripplemode, RMX/80, RUPI, Seamless, SLD, UPI, and VLSiCEL, and the combination of ICE, iCS, iRMX, iSBC, iSBX, MCS, or UPI and a numerical suffix.

MDS is an ordering code only and is not used as a product name or trademark. MDS® is a registered trademark of Mohawk Data Sciences Corporation.

*MULTIBUS is a patented Intel bus.

Additional copies of this manual or other Intel literature may be obtained from:

Intel Corporation
Literature Distribution
Mail Stop SC6-59
3065 Bowers Avenue
Santa Clara, CA 95051

©INTEL CORPORATION 1985

Table of Contents

NUMERIC INDEX vi

CHAPTER 1
OVERVIEW
Introduction 1-1

CHAPTER 2
8080, 8085 MICROPROCESSORS
DATA SHEETS
8080A/8080A-1/8080A-2 8-Bit N-Channel Microprocessor 2-1
8085AH/8085AH-2/8085AH-1 8-Bit HMOS Microprocessor 2-10
8155H/8156H/8155H-2/8156H-2 2048-Bit Static HMOS RAM with I/O Ports and Timer 2-26
8185/8185-2 1024 x 8-Bit Static RAM for MCS®-85 2-38
8224 Clock Generator and Driver for 8080A CPU 2-43
8228/8238 System Controller and Bus Driver for 8080A CPU 2-48
8237A/8237A-4/8237A-5 High Performance Programmable DMA Controller 2-52
82C37A-5 CHMOS High Performance Programmable DMA Controller 2-67
8257/8257-5 Programmable DMA Controller 2-78
8259A/8259A-2/8259A-8 Programmable Interrupt Controller 2-95
82C59A-2 CHMOS Programmable Interrupt Controller 2-113
8755A/8755A-2 16, 384-Bit EPROM with I/O 2-133
APPLICATION NOTES
AP-59 Using the 8259A Programmable Interrupt Controller 2-144

CHAPTER 3
8086, 8088, 80186, 80188 MICROPROCESSCORS
DATA SHEETS
8086 16-Bit HMOS Microprocessor 3-1
80C86/80C86-2 16-Bit CHMOS Microprocessor 3-25
80186 High Integration 16-Bit Microprocessor 3-52
iAPX 88/10 8-Bit HMOS Microprocessor 3-106
80C88/80C88-2 8-Bit CHMOS Microprocessor 3-133
80188 High Integration 8-Bit Microprocessor 3-162
8087/8087-2/8087-1 Numeric Data Coprocessor 3-218
8282/8283 Octal Latch 3-241
8284A/8284A-1 Clock Generator and Driver for iAPX 86, 88 Processors 3-246
82C84A/82C84A-5 CHMOS Clock Generator and Driver for 80C86, 80C88 Processors 3-254
8286/8287 Octal Bus Transceiver 3-263
8288 Bus Controller for iAPX 86, 88 Processors 3-268
82C88 CHMOS Bus Controller for 80C86, 80C88 Processors 3-275
82188 Integrated Bus Controller for iAPX 86, 88, 186, 188 Processors 3-283
8289/8289-1 Bus Arbiter 3-299
APPLICATION NOTES
AP-67 8086 System Design 3-310
AP-113 Getting Started with the Numeric Data Processor 3-373
AP-186 Introduction to the 80186 3-435

CHAPTER 4
80286 MICROPROCESSORS
DATA SHEETS
iAPX 286/10 High Performance Microprocessor
with Memory Management and Protection 4-1
80287 80-Bit HMOS Numeric Processor Extension 4-56
82258 Advanced Direct Memory Access Coprocessor 4-82
82284 Clock Generator and Ready Interface for iAPX 286 Processors 4-139
82288 Bus Controller for iAPX 286 Processors 4-148
82289 Bus Arbiter for iAPX 286 Processor Family 4-167

CHAPTER 5
80386 MICROPROCESSORS
DATA SHEETS
80386 High Performance Microprocessor with Integrated Memory Management 5-1
82384 Clock Generator and Reset Interface for 80386 Processors 5-2

—VOLUME 2—

CHAPTER 6

MEMORY CONTROLLERS

DATA SHEETS

8202A Dynamic RAM Controller 6-1
8203 64K Dynamic RAM Controller 6-15
8206/8206-2 Error Detection and Correction Unit 6-30
8207 Dual-Port Dynamic RAM Controller 6-52
8208 Dynamic RAM Controller 6-98
82C08 CHMOS Dynamic RAM Controller 6-121

USERS MANUAL

Introduction 6-151
Programming the 8207 6-152
RAM Interface 6-157
Microprocessor Interfaces 6-166
8207 with ECC (8206) 6-174
Appendix 6-177

APPLICATION NOTES

AP-97A Interfacing Dynamic RAM to iAPX 86/88 Using the 8202A & 8203 6-181
AP-141 8203/8206/2164A Memory Design 6-217
AP-167 Interfacing the 8207 Dynamic RAM Controller to the iAPX 186 6-223
AP-168 Interfacing the 8207 Advanced Dynamic RAM Controller to the iAPX 286 6-228

ARTICLE REPRINTS

AR-364 FAE News 1/84 "8208 with 186" 6-235
AR-231 Dynamic RAM Controller Orchestrates Memory Systems 6-246

SUPPORT PERIPHERALS

DATA SHEETS

8231A Arithmetic Processing Unit 6-253
8253/8253-5 Programmable Interval Timer 6-263
8254 Programmable Interval Timer 6-274
82C54 CHMOS Programmable Interval Timer 6-290
8255A/8255A-5 Programmable Peripheral Interface 6-307
82C55A CHMOS Programmable Peripheral Interface 6-328
8256AH Multifunction Microprocessor Support Controller 6-351
8279/8279-5 Programmable Keyboard/Display Interface 6-374

APPLICATION NOTES

AP-153 Designing with the 8256 6-386
AP-183 8256AH Application Note 6-461

FLOPPY DISK CONTROLLERS

DATA SHEETS

8272A Single/Double Density Floppy Disk Controller 6-478

APPLICATION NOTES

AP-116 An Intelligent Data Base System Using the 8272 6-497
AP-121 Software Design and Implementation of Floppy Disk Systems 6-538

HARD DISK CONTROLLERS

DATA SHEETS

82062 Winchester Disk Controller 6-608
82064 Winchester Disk Controller with On-Chip Error Detection and Correction 6-635

APPLICATION NOTES

AP-182 Multimode Winchester Controller Using the 82062 6-667

UNIVERSAL PERIPHERAL INTERFACE SLAVE MICROCONTROLLERS

DATA SHEETS

UPI-452 Slave Microcontroller (8051) 6-729
UPI-41 8-Bit Slave Microcontroller 6-768
UPI-42 8-Bit Slave Microcontroller 6-780
8243 MCS-48 Input/Output Expander 6-799

UPI-41/42 USERS MANUAL
Introduction 6-805
Functional Description 6-810
Instruction Set 6-827
Single-Step, Programming, and Power-Down Modes 6-854
System Operation 6-859
Applications 6-865
AP-161 or 61 Complex Peripheral Control with the UPI-42 6-939
AP-90 An 8741A/8041A Digital Cassette Controller 6-995
APPLICATION NOTES
Applications Using the 8042 UPI™ Microcontroller 6-1003
SYSTEM SUPPORT
ICE-42 8042 In-Circuit Emulator 6-1007
MCS-48 Diskette-Based Software Support Package 6-1015
iUP-200/iUP-201 Universal PROM Programmers 6-1017

CHAPTER 7
ALPHANUMERIC TERMINAL CONTROLLERS
DATA SHEETS
8275H Programmable CRT Controller 7-1
8276H Small System CRT Controller 7-25
APPLICATION NOTES
AP-62 A Low Cost CRT Terminal Using the 8275 7-42
ARTICLE REPRINTS
AR-178 A Low Cost CRT Terminal Does More with Less 7-84
GRAPHICS DISPLAY PRODUCTS
DATA SHEETS
82720 Graphics Display Controller 7-91
ARTICLE REPRINTS
AR-255 Dedicated VLSI Chip Lightens Graphic Display Design Load 7-128
AR-298 Graphics Chip Makes Low Cost High Resolution, Color Displays Possible 7-136
TEXT PROCESSING PRODUCTS
DATA SHEETS
82730 Text Coprocessor 7-143
ARCHITECTURAL OVERVIEW
The 82786 CHMOS Graphics Coprocessor 7-187
ARTICLE REPRINTS
AR-305 Text Coprocessor Brings Quality to CRT Displays 7-205
AR-297 VLSI Coprocessor Delivers High Quality Displays 7-213
AR-296 Mighty Chips 7-216

CHAPTER 8
ERASABLE/PROGRAMMABLE LOGIC DEVICES
DATA SHEETS
5C121 1200 Gate CHMOS H-Series Eraseable/Programmable Logic Device 8-1
5C060 600 Gate CHMOS H-Series Erasable/Programmable Logic Device 8-15

Numeric Index

5C121 1200 Gate CHMOS H-Series Eraseable/Programmable Logic Device 8-1
5C060 600 Gate CHMOS H-Series Eraseable/Programmable Logic Device 8-15
80186 (iAPX 186) High Integration 16-Bit Microprocessor 3-52, 3-435
80188 (iAPX 188) High Integration 8-Bit Microprocessor 3-162
80286 (iAPX 286/10) High Performance Microprocessor with Memory Management and Protection 4-1, 6-228, 6-247
80287 80-Bit HMOS Numeric Processor Extension 4-56
80386 High Performance Microprocessor with Integrated Memory Management 5-1
8041A/8641A/8741A Universal Peripheral Interface
8-Bit Slave Micro Controller 6-768, 6-805, 6-994
8042/8742 Universal Peripheral Interface
8-Bit Slave Micro Controller 6-780, 6-805, 6-939, 6-1002, 6-1006
80452/83452/87452 Universal Peripheral Interface 8-Bit Slave Micro Controller ... 6-729, 6-805
8080A/8080A-1/8080A-2, 8-Bit N-Channel Microprocessor 2-1
8085AH/8085AH-2/8085AH-1 8-Bit HMOS Microprocessors 2-10
8086 (iAPX 86/10) 16-Bit HMOS Microprocessor 3-1, 3-310, 6-181
80C86/80C86-2 16-Bit Microprocessor 3-25
8087/8087-2/8087-1 Numeric Data Coprocessor 3-218, 3-373
8088 (iAPX 88/10) 8-Bit HMOS Microprocessor 3-106, 6-181
80C88/80C88-2 8-Bit CHMOS Microprocessor 3-133
8155H/8156H/8155H-2/8156H-2 2048-Bit Static HMOS RAM with I/O Ports and Timer 2-26
8185/8185-2 1024 x 8-Bit Static RAM for MCS®-85 2-38
8202A Dynamic RAM Controller 6-1, 6-181
8203 64K Dynamic RAM Controller 6-15, 6-181, 6-217
8205 High Speed 1 out of 8 Binary Decoder .....
8206 Error Detection and Correction Unit 6-30, 6-217, 6-247
82062 Winchester Disk Controller 6-608, 6-667
82064 Winchester Disk Controller with On-Chip Error Detection and Correction 6-635
8207 Dual-Port Dynamic RAM Controller 6-52, 6-150, 6-223, 6-228, 6-247
8208 Dynamic RAM Controller 6-98, 6-235
82C08 Dynamic RAM Controller 6-121
82188 Integrated Bus Controller for iAPX 86, 88, 186, 188 Processors 3-283
8224 Clock Generator And Driver for 8080A CPU 2-43
82258 Advanced Direct Memory Access Coprocessor 4-82
8228/8238 System Controller and Bus Driver for 8080A CPU 2-48
82284 Clock Generator and Ready Interface for iAPX 286 Processors 4-139
82288 Bus Controller for iAPX 286 Processors 4-148
82289 Bus Arbiter for iAPX 286 Processor Family 4-167
8231A Arithmetic Processing Unit 6-253
8237A/8237A-4/8237A-5 High Performance Programmable DMA Controller 2-52
82C37A-5 CHMOS High Performance Programmable DMA Controller 2-67
82384 Clock Generator And Reset Interface for 80386 Processors 5-2
8243 MCS-48 Input/Output Expander 6-799, 6-805

8253/8253-5 Programmable Interval Timer 6-263
8254 Programmable Interval Timer 6-274
82C54 CHMOS Programmable Interval Timer 6-290
8255A/8255A-5 Programmable Peripheral Interface 6-307
82C55A CHMOS Programmable Peripheral Interface 6-328
8256AH Multifunction Microprocessor Support Controller 6-357, 6-386, 6-461
8257/8257-5 Programmable DMA Controller 2-78
8259A/8259A-2/8259A-8 Programmable Interrupt Controller 2-95, 2-144
82C59A-2 CHMOS Programmable Interrupt Controller 2-113
8272A Single/Double Density Floppy Disk Controller 6-478, 6-497, 6-538
82720 Graphics Display Controller 7-91, 7-128, 7-136, 7-205, 7-213, 7-216
82730 Text Coprocessor 7-136, 7-143, 7-205, 7-213, 7-216.....
8275H Programmable CRT Controller 7-1, 7-42
8276H Small System CRT Controller 7-25, 7-84
82786 7-187
8279/8279-5 Programmable Keyboard/Display Interface 6-374
8282/8283 Octal Latch 3-241
8284A/8284A-1 Clock Generator and Driver for iAPX 86, 88 Processors 3-246
82C84A/82C84A-5 CHMOS Clock Generator And Driver
For 80C86, 80C88 Processors 3-254
8286/8287 Octal Bus Transceiver 3-263.....
8288 Bus Controller for iAPX 86, 88 Processors 3-268
82C88 CHMOS Bus Controller for 80C86, 80C88 Processors 3-275
8289/8989-1 Bus Arbiter 3-299
8755A/8755A-2 16,384-Bit EPROM with I/O 2-133

CUSTOMER SUPPORT

CUSTOMER SUPPORT

Customer Support is Intel's complete support service that provides Intel customers with Customer Training, Software Support and Hardware Support.

After a customer purchases any system hardware or software product, service and support become major factors in determining whether that product will continue to meet a customer's expectations. Such support requires an international support organization and a breadth of programs to meet a variety of customer needs. Intel's extensive customer support includes factory repair services as well as worldwide field service offices providing hardware repair services, software support services and customer training classes.

HARDWARE SUPPORT

Hardware Support Services provides maintenance on Intel supported products at board and system level. Both field and factory services are offered. Services include several types of field maintenance agreements, installation and warranty services, hourly contracted services (factory return for repair) and specially negotiated support agreements for system integrators and large volume end-users having unique service requirements. For more information contact your local Intel Sales Office.

SOFTWARE SUPPORT

Software Support Service provides maintenance on software packages via software support contracts which include subscription services, information phone support, and updates. Consulting services can be arranged for on-site assistance at the customer's location for both short-term and long-term needs. For complex products such as NDS II or I²ICE, orientation/installation packages are available through membership in Insite User's Library, where customer-submitted programs are catalogued and made available for a minimum fee to members. For more information contact your local Intel Sales Office.

CUSTOMER TRAINING

Customer Training provides workshops at customer sites (by agreement) and on a regularly scheduled basis at Intel's facilities. Intel offers a breadth of workshops on microprocessors, operating systems and programming languages, etc. For more information on these classes contact the Training Center nearest you.

TRAINING CENTER LOCATIONS

To obtain a complete catalog of our workshops, call the nearest Training Center in your area.

Boston	(617) 692-1000	London	(0793) 696-000
Chicago	(312) 310-5700	Munich	(089) 5389-1
San Francisco	(415) 940-7800	Paris	(01) 687-22-21
Washington, D.C.	(301) 474-2878	Stockholm	(468) 734-01-00
Israel	(972) 349-491-099	Milan	39-2-82-44-071
Tokyo	03-437-6611	Benelux (Rotterdam)	(10) 21-23-77
Osaka (Call Tokyo)	03-437-6611	Copenhagen	(1) 198-033
Toronto, Canada	(416) 675-2105	Hong Kong	5-215311-7

INTRODUCTION

Intel microprocessors and peripherals provide a complete solution in increasingly complex application environments. Quite often, a single peripheral device will replace anywhere from 20 to 100 TTL devices (and the associated design time that goes with them).

Built-in functions and standard Intel microprocessor/peripheral interface deliver very real *time* and *performance* advantages to the designer of microprocessor-based systems.

REDUCED TIME TO MARKET

When you can purchase an off-the-shelf solution that replaces a number of discrete devices, you're also replacing all the design, testing, and debug *time* that goes with them.

INCREASED RELIABILITY

At Intel, the rate of failure for devices is carefully tracked. Highest reliability is a tangible goal that translates to higher reliability for your product, reduced downtime, and reduced repair costs. And as more and more functions are intergrated on a single VLSI device, the resulting system requires less power, produces less heat, and requires fewer mechanical connections—again resulting in greater system reliability.

LOWER PRODUCTION COST

By minimizing design time, increasing reliability, and replacing numerous parts, microprocessor and peripheral solutions can contribute dramatically to lower product costs.

HIGHER SYSTEM PERFORMANCE

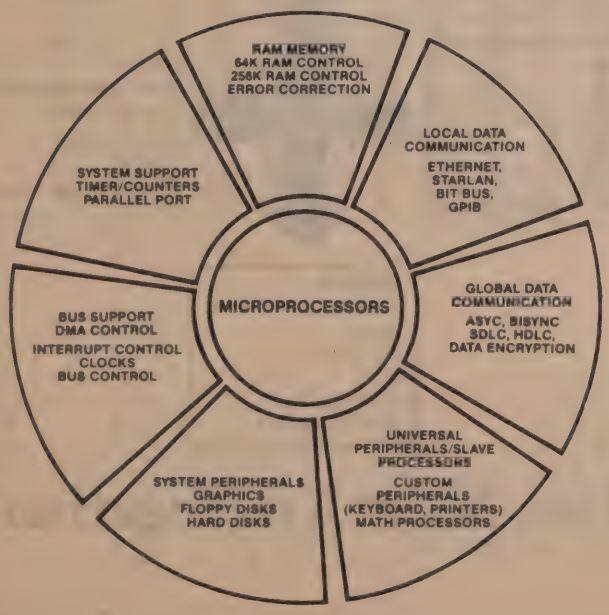
Intel microprocessors and peripherals provide the highest system performance for the demands of today's (and tomorrow's) microprocessor-based applications. For example, the 80386 32 bit offers the highest performance for multitasking, multiuser systems. Intel's peripheral products have been designed with the future in mind. They support all of Intel's 8, 16 and 32 bit processors.

HOW TO USE THE GUIDE

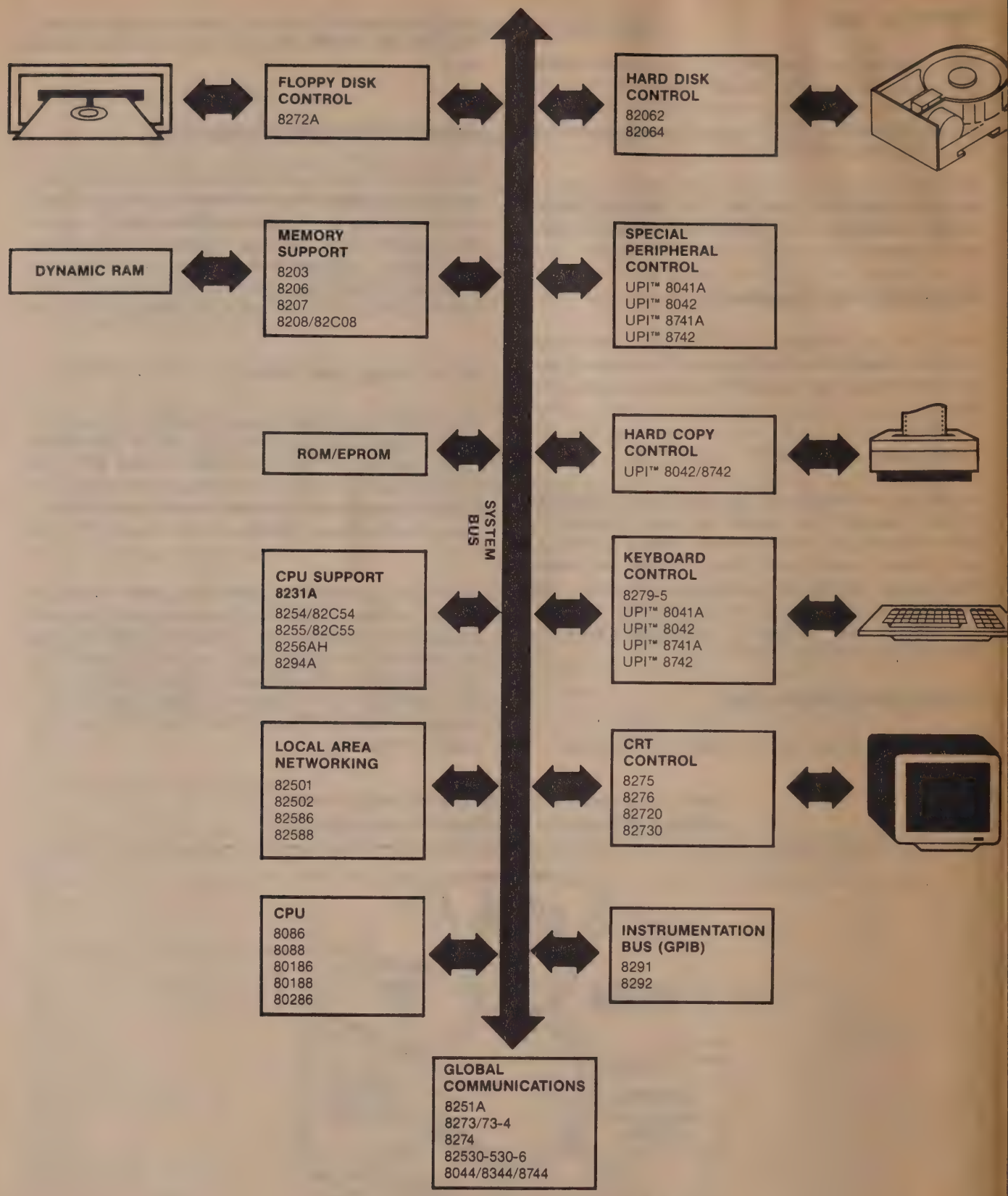
The following application guide illustrates the range of microprocessors and peripherals that can be used for the applictions in the vertical column of the left. The peripherals are grouped by the I/O function they control. CRT datacommunication, universal (user programmable), mass storage dynamic RAM controllers, and CPU/bus support.

An "X" in a horizontal application row indicates a potential peripheral or CPU, depending upon the features desired. For example, a conversational terminal could use either of the three display controllers, depending upon features like the number of characters per row or font capability. A "Y" indicates a likely candidate, for example, the 8272A Floppy Disk Controller in a small business computer.

The Intel microprocessor and peripherals family provides a broad range of time-saving, high performance solutions.



Intel's Microsystem Components Kit Solution



Get Your Kit Together!

©Intel 1985

September 1985
Order Number: 230664-004

APPLICATION CHART

POTENTIAL APPLICATION X—TYPICAL APPLICATION Y

APPLICATION	MICROPROCESSORS					DISPLAY			DATA COMMUNICATIONS								UPI	DISK		DRAM CONTROL				SUPPORT			
	8088	8086	80188	80186	80286	8275/76	82720	82730/731*	8251A	8273*	8274	8291A/92/94*	82530	82588*	82586/501/502*	8044/8744*	8042/8742	8272A	82062/064	8203*	8206	8207*	8208*/C08*	8254/C54	8255/C55	8256AH	8279/8279-5
PERIPHERALS																											
Printers	X		X	Y			X	X	X			X				X	X							X	X	X	
Plotters	X	X	X	Y					X			X				X	X								X		
Keyboards			X	Y					X							X	X								X		X
MASS STORAGE																											
Hard Disk	X	X	Y	Y															X						X		
Mini Winchester	X		Y	Y															Y								
Tape			X	Y												X	X								X		
Cassette			Y	Y													X								X		
Floppy/Mini			Y	Y														Y									
COMMUNICATIONS																											
PBX			X	X	Y					X	X		X			X				X	X	X	X	X	X	X	
LANS	X	X	Y	Y							X		X	X	Y	X								X			
Modems									X		X		X			X	X										X
Bisync									X		X			X										X			
SDLC/HDLC										X	X		X			X								X			
Serial Backplane										X	X		X		X	X											
Central Office		X		X	Y					X	X		X			X					X	X	X	X		X	
Network Control		Y		X	Y						X		X	X	X	X								X			
OFFICE/BUS																											
Copier/FAX	X		X	Y								X			X	X	X									X	X
Wordprocessor	X	X	X	X	Y		X	Y	X							X		Y		X	X	X	X	Y	Y	X	
Typewriter			X													X	X								Y		
Electronic Mail		X	X	X			X	Y			X		X	X	X						X						
Transaction System		Y	X	X	X				X							X								Y	X	X	
Data Entry	X	X	X	X		X	X	X	X							X	X						X	Y		X	X
COMPUTERS																											
SM Bus Computer		X	Y	X	X	X	Y	Y	X		X		X	X	X	X	X	Y	Y	X	X	X	X	Y	Y	X	
PC	Y	X		X	X	X	Y	Y	X		X	X	X	X	X	X	X	Y	Y	X	X	X	X	Y	Y	X	
Portable PC				X					X								X	Y		Y			X	Y	Y	X	
Home Computer	X	X	Y	X	X		X		X								X	Y		X		X	X	Y	Y	X	

APPLICATION	MICROPROCESSORS					DISPLAY			DATA COMMUNICATIONS								UPI	DISK		DRAM CONTROL				SUPPORT			
	8088	8086	80188	80186	80286	8275/76	82720	82730/731*	8251A	8273*	8274	8291A/92/94*	82530	82588*	82586/501/502*	8044/8744*	8042/8742	8272A	82062/064	8203*	8206	8207*	8208*/C08*	8254/C54	8255/C55	8256AH	8279/8279-5
TERMINALS																											
Conversational			Y			X	X	X	X	X												X	X	Y		X	
Graphics CRT		Y		Y	Y		Y	Y	X		X		X		X			X	X	X	X	X	X	Y	Y	X	
Editing	X	X	X	X	X		Y	Y	X		X		X		X	X		X		X		X	X	Y	Y	Y	
Intelligent	X	X	Y	Y				Y	X		X		X	X	X			X	X			X	X	Y	Y	X	
Videotex	X	X	X	X			X	X	X													X	X	Y	Y		
Printing, Laser, Impact	X	X	X	X			X	X	X						X	X	X					X	X	Y	Y		
Portable	X	X	Y				X	X	X									X		X		X	X	Y	Y		
INDUSTRIAL AUTO																											
Robotics			Y	Y	X						X		X			Y	X				X				Y		X
Network	X	X	X	X	X					X	X		X	X	X	Y	X				X						
Numeric Control		X	X	X	X		Y			X	X		X		X	Y	X				X						X
Process Control	X	X	X	Y	X		Y			X	X		X		X	Y	X			X	X		X	X	X		X
Instrumentation	X	X	X	X			Y					X				Y	X				X			X	X		X
Aviation/Navigation		X	X	X	X												X				X			X	X		X
INDUSTRIAL/DATA ACQ.																											
Laboratory Instrumentation	X	X	Y	X	X							Y				Y	X					X	X		X		X
Source Data	X		Y													Y						X	X		X		X
Auto Test	X	X	Y	X	Y											Y						X	X		X		X
Medical	X	X	Y	X	X		Y								X	Y	X				X	X	X		X		X
Test Instrumentation	X	X	Y	X	X						X		X		X	Y	X					X	X		X		X
Security		X	Y	X										X		Y	X				X	X	X		X		X
COMMERCIAL DATA PROCESSING																											
POS Terminal		X	X	Y		X	X	X	X		X		X		X	X	X			X	X	X	X	X	X	X	
Financial Transfer		X	X	Y		X	X	X	X					X		Y					X	X	X		X	X	
Automatic Teller		X	X	Y		X	X	X	X							Y	X				X	X	X		X	X	X
Document Processing	X	X	X	Y	X	X	X	X	X							Y						X	X			X	
WORKSTATIONS																											
Office	X	X	X	Y	X		Y	Y	X		X		X	X	X	X	X	Y	Y	X	X	X	X	X	Y	X	
Engineering	X	X		Y	X		Y	Y	X		X		X	X	X		X	Y	Y	X	X	X	X	X	Y	X	
CAD		X		Y	Y		Y	Y	X		X		X		X		X	Y	Y	X	X	X	X	X	Y		
MINI MAINFRAME																											
Processor & Control Store		X		Y	Y	Y					X		X								X	X					
Database Subsystems		X		Y	X	X								X							X	X					
I/O Subsystem			Y	Y	Y						X		X			X						X					
Comm. Subsystem		X		Y	Y				X		X		X	X		X						X					

*Single Source Product

Peripherals

6

8202A
DYNAMIC RAM CONTROLLER

- **Provides All Signals Necessary to Control 2117, or 2118 Dynamic Memories**
- **Directly Addresses and Drives Up to 64K Bytes Without External Drivers**
- **Provides Address Multiplexing and Strobes**
- **Provides a Refresh Timer and a Refresh Counter**
- **Refresh Cycles May be Internally or Externally Requested**
- **Provides Transparent Refresh Capability**
- **Fully Compatible with Intel® 8080A, 8085A, iAPX 88, and iAPX 86 Family Microprocessors**
- **Decodes CPU Status for Advanced Read Capability with the 8202A-1 or 8202A-3**
- **Provides System Acknowledge and Transfer Acknowledge Signals**
- **Internal Clock Capability with the 8202A-1 or 8202A-3**

The Intel® 8202A is a Dynamic Ram System Controller designed to provide all signals necessary to use 2117 or 2118 Dynamic RAMs in microcomputer systems. The 8202A provides multiplexed addresses and address strobes, as well as refresh/access arbitration. The 8202A-1 or 8202A-3 support an internal crystal oscillator.

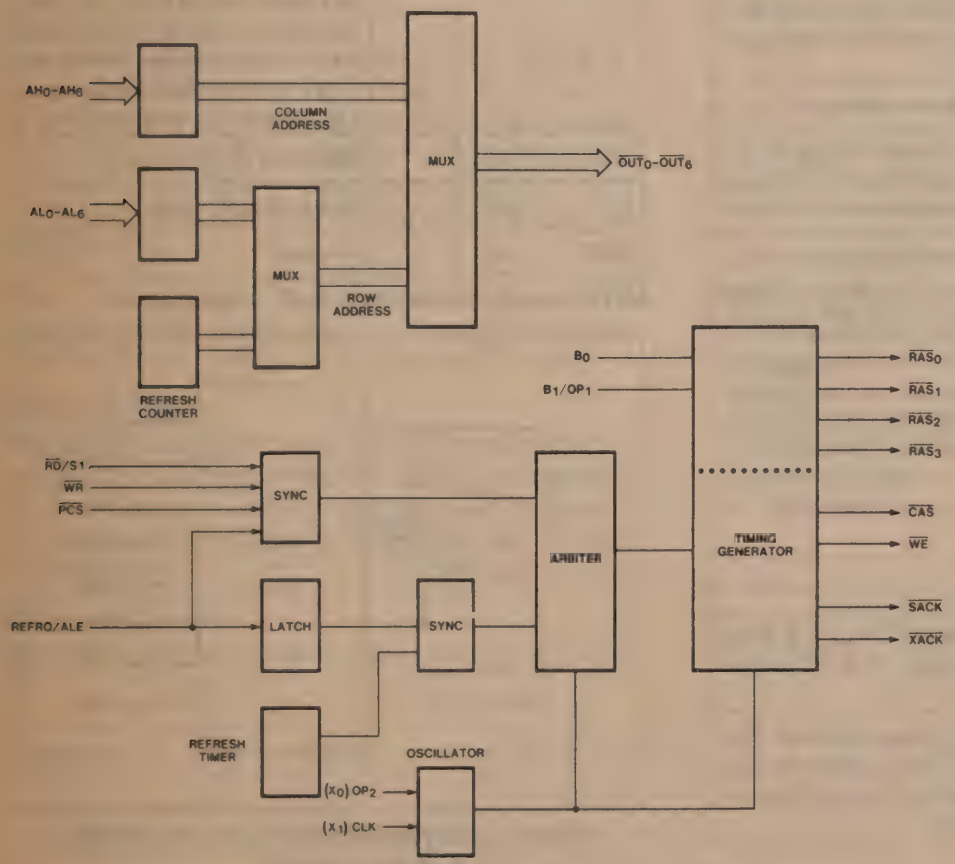


Figure 1. 8202A Block Diagram

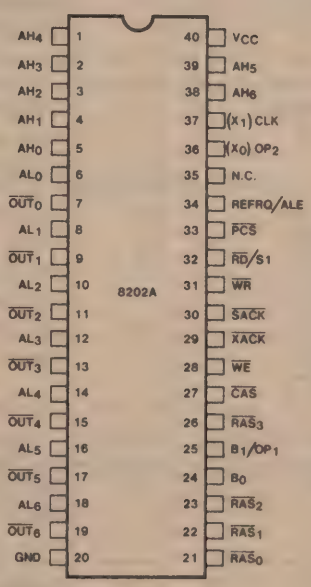


Figure 2. Pin Configuration

205215-001

Table 1. Pin Descriptions

Symbol	Pin No.	Type	Name and Function
AL_0 AL_1 AL_2 AL_3 AL_4 AL_5 AL_6/	6 8 10 12 14 16 18	I I I I I I I	**Address Low:** CPU address inputs used to generate memory row address.
AH_0 AH_1 AH_2 AH_3 AH_4 AH_5 AH_6	5 4 3 2 1 39 38	I I I I I I I	**Address High:** CPU address inputs used to generate memory column address.
BO B_1/OP_1	24 25	I I	**Bank Select Inputs:** Used to gate the appropriate RAS_0-RAS_3 output for a memory cycle. B_1/OP_1 option used to select the Advanced Read Mode.
$\overline{PCS}$	33	I	**Protected Chip Select:** Used to enable the memory read and write inputs. Once a cycle is started, it will not abort even if $\overline{PCS}$ goes inactive before cycle completion.
$\overline{WR}$	31	I	**Memory Write Request.**
$\overline{RD}$/S1	32	I	**Memory Read Request:** S1 function used in Advanced Read mode selected by OP_1 (pin 25).
REFRQ/ALE	34	I	**External Refresh Request:** ALE function used in Advanced Read mode, selected by OP_1 (pin 25).
$\overline{OUT}_0$ $\overline{OUT}_1$ $\overline{OUT}_2$ $\overline{OUT}_3$ $\overline{OUT}_4$ $\overline{OUT}_5$ $\overline{OUT}_6$	7 9 11 13 15 17 19	O O O O O O O	**Output of the Multiplexer:** These outputs are designed to drive the addresses of the Dynamic RAM array. (Note that the $\overline{OUT}_{0-6}$ pins do not require inverters or drivers for proper operation.)
$\overline{WE}$	28	O	**Write Enable:** Drives the Write Enable inputs of the Dynamic RAM array.
$\overline{CAS}$	27	O	**Column Address Strobe:** This output is used to latch the Column Address into the Dynamic RAM array.
$\overline{RAS}_0$ $\overline{RAS}_1$ $\overline{RAS}_2$ $\overline{RAS}_3$	21 22 23 26	O O O O	**Row Address Strobe:** Used to latch the Row Address into the bank of dynamic RAMs, selected by the 8202A Bank Select pins (B_0, B_1/OP_1).
$\overline{XACK}$	29	O	**Transfer Acknowledge:** This output is a strobe indicating valid data during a read cycle or data written during a write cycle. $\overline{XACK}$ can be used to latch valid data from the RAM array.
$\overline{SACK}$	30	O	**System Acknowledge:** This output indicates the beginning of a memory access cycle. It can be used as an advanced transfer acknowledge to eliminate wait states. (Note: If a memory access request is made during a refresh cycle, $\overline{SACK}$ is delayed until $\overline{XACK}$ in the memory access cycle).
(X_0) OP_2 (X_1) CLK	36 37	I/O I/O	**Oscillator Inputs:** These inputs are designed for a quartz crystal to control the frequency of the oscillator. If X_0/OP_2 is connected to a 1KΩ resistor pulled to +12V then X_1/CLK becomes a TTL input for an external clock.
N.C.	35		Reserved for future use.
V_{CC}	40		**Power Supply:** +5V.
GND	20		**Ground.**

NOTE: Crystal mode for the 8202A-1 or 8202A-3 only.

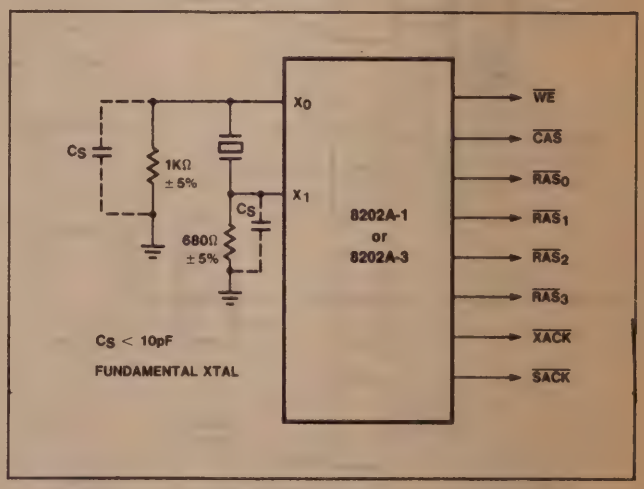


Figure 3. Crystal Operation for the 8202A-1 and the 8202A-3

Functional Description

The 8202A provides a complete dynamic RAM controller for microprocessor systems as well as expansion memory boards. All of the necessary control signals are provided for 2117 and 2118 dynamic RAMs.

All 8202A timing is generated from a single reference clock. This clock is provided via an external oscillator or an on chip crystal oscillator. All output signal transitions are synchronous with respect to this clock reference, except for the CPU handshake signals $\overline{SACK}$ and $\overline{XACK}$ (trailing edge).

CPU memory requests normally use the $\overline{RD}$ and $\overline{WR}$ inputs. The advanced READ mode allows ALE and S1 to be used in place of the $\overline{RD}$ input.

Failsafe refresh is provided via an internal refresh timer which generates internal refresh requests. Refresh requests can also be generated via the REFRQ input.

An on-chip synchronizer/arbiter prevents memory and refresh requests from affecting a cycle in progress. The READ, WRITE, and external REFRESH requests may be asynchronous to the 8202A clock; on-chip logic will synchronize the requests, and the arbiter will decide if the requests should be delayed, pending completion of a cycle in progress.

Option Selection

The 8202A has two strapping options. When OP_1 is selected (16K mode only), pin 32 changes from a RD input to an S1 input, and pin 34 changes from a REFREQ input to an ALE input. See "Refresh Cycles" and "Read Cycles" for more detail. OP_1 is selected by tying pin 25 to +12V through a 5.1K ohm resistor on the 8202A-1 or 8202A-3 only.

When OP_2 is selected, by connecting pin 36 to +12V through a 1K ohm resistor, pin 37 changes from a crystal input (X_1) to the CLK input for an external TTL clock.

Refresh Timer

The refresh timer is used to monitor the time since the last refresh cycle occurred. When the appropriate amount of time has elapsed, the refresh timer will request a refresh cycle. External refresh requests will reset the refresh timer.

Refresh Counter

The refresh counter is used to sequentially refresh all of the memory's rows. The 8-bit counter is incremented after every refresh cycle.

Address Multiplexer

The address multiplexer takes the address inputs and the refresh counter outputs, and gates them onto the address outputs at the appropriate time. The address outputs, in conjunction with the $\overline{RAS}$ and $\overline{CAS}$ outputs, determine the address used by the dynamic RAMs for read, write, and refresh cycles. During the first part of a read or write cycle, AL_0–AL_6 are gated to $\overline{OUT}_0$–$\overline{OUT}_6$, then AH_0–AH_6 are gated to the address outputs.

During a refresh cycle, the refresh counter is gated onto the address outputs. All refresh cycles are RAS-only refresh ($\overline{CAS}$ inactive, $\overline{RAS}$ active).

To minimize buffer delay, the information on the address outputs is inverted from that on the address inputs.

$\overline{OUT}_0$–$\overline{OUT}_6$ do not need inverters or buffers unless additional drive is required.

Synchronizer/Arbiter

The 8202A has three inputs, REFRQ/ALE (pin 34), $\overline{RD}$ (pin 32) and $\overline{WR}$ (pin 31). The $\overline{RD}$ and $\overline{WR}$ inputs allow an external CPU to request a memory read or write cycle, respectively. The REFRQ/ALE allows refresh requests to be requested external to the 8202A.

All three of these inputs may be asynchronous with respect to the 8202A's clock. The arbiter will resolve conflicts between refresh and memory requests, for both pending cycles and cycles in progress. Read and write requests will be given priority over refresh requests.

System Operation

The 8202A is always in one of the following states:

a) IDLE
b) TEST Cycle
c) REFRESH Cycle
d) READ Cycle
e) WRITE Cycle

The 8202A is normally in the IDLE state. Whenever one of the other cycles is requested, the 8202A will leave the IDLE state to perform the desired cycle. If no other cycles are pending, the 8202A will return to the IDLE state.

Description	Pin #	Normal Function	Option Function
B1/OP1	25	Bank (RAS) Select	Advanced-Read Mode (see text)
X_0/OP_2	36	Crystal Oscillator (8202A-1 or 8202A-3)	External Oscillator

Figure 4. 8202A Option Selection

Test Cycle

The TEST Cycle is used to check operation of several 8202A internal functions. TEST cycles are requested by activating the $\overline{RD}$ and $\overline{WR}$ inputs, independent of $\overline{PCS}$. The TEST Cycle will reset the refresh address counter. It will perform a WRITE Cycle if $\overline{PCS}$ is low. The TEST Cycle should not be used in normal system operation, since it would affect the dynamic RAM refresh.

Refresh Cycles

The 8202A has two ways of providing dynamic RAM refresh:

1) Internal (failsafe) refresh
2) External (hidden) refresh

Both types of 8202A refresh cycles activate all of the $\overline{RAS}$ outputs, while $\overline{CAS}$, $\overline{WE}$, $\overline{SACK}$, and $\overline{XACK}$ remain inactive.

Internal refresh is generated by the on-chip refresh timer. The timer uses the 8202A clock to ensure that refresh of all rows of the dynamic RAM occurs every 2 milliseconds. If REFRQ is inactive, the refresh timer will request a refresh cycle every 10-16 microseconds.

External refresh is requested via the REFRQ input (pin 34). External refresh control is not available when the Advanced-Read mode is selected. External refresh requests are latched, then synchronized to the 8202A clock.

The arbiter will allow the refresh request to start a refresh cycle only if the 8202A is not in the middle of a cycle.

Simultaneous memory request and external refresh request will result in the memory request being honored first. This 8202A characteristic can be used to "hide" refresh cycles during system operation. A circuit similar to Figure 5 can be used to decode the CPU's instruction fetch status to generate an external refresh request. The refresh request is latched while the 8202A performs the instruction fetch; the refresh cycle will start immediately after the memory cycle is completed, even if the $\overline{RD}$ input has not gone inactive. If the CPU's instruction decode time is long enough, the 8202A can complete the refresh cycle before the next memory request is generated.

Certain system configurations require complete external refresh requests. If external refresh is requested faster than the minimum internal refresh timer (t_{REF}), then, in effect, all refresh cycles will be caused by the external refresh request, and the internal refresh timer will never generate a refresh request.

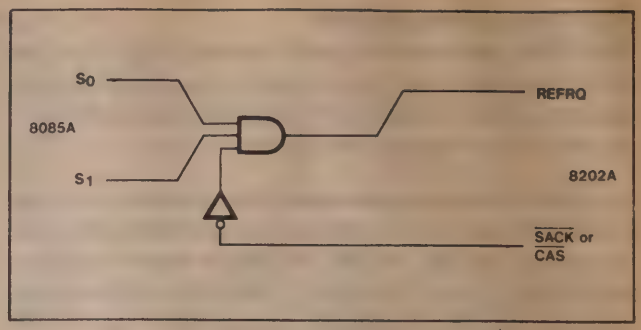


Figure 5. Hidden Refresh

Read Cycles

The 8202A can accept two different types of memory Read requests:

1) Normal Read, via the $\overline{RD}$ input
2) Advanced Read, using the S1 and ALE inputs

The user can select the desired Read request configuration via the B1/OP1 hardware strapping option on pin 25.

	Normal Read	Advanced Read
Pin 25	B1 input	+12 Volt Option
Pin 32	RD input	S1 input
Pin 34	REFRQ input	ALE input
# RAM banks	4 ($\overline{RAS}_{0-3}$)	2 ($\overline{RAS}_{2-3}$)
Ext. Refresh Req.	Yes	No

Figure 6. 8202A Read Options

Normal Reads are requested by activating the $\overline{RD}$ input, and keeping it active until the 8202A responds with an $\overline{XACK}$ pulse. The $\overline{RD}$ input can go inactive as soon as the command hold time (t_{CHS}) is met.

Advanced Read cycles are requested by pulsing ALE while S1 is active; if S1 is inactive (low) ALE is ignored. Advanced Read timing is similiar to Normal Read timing, except the falling edge of ALE is used as the cycle start reference.

If a Read cycle is requested while a refresh cycle is in progress, then the 8202A will set the internal delayed-SACK latch. When the Read cycle is eventually started, the 8202A will delay the active $\overline{SACK}$ transition until $\overline{XACK}$ goes active, as shown in the AC timing diagrams. This delay was designed to compensate for the CPU's READY setup and hold times. The delayed–SACK latch is cleared after every READ cycle.

Based on system requirements, either $\overline{SACK}$ or $\overline{XACK}$ can be used to generate the CPU READY signal. $\overline{XACK}$ will

normally be used; if the CPU can tolerate an advanced READY, then $\overline{SACK}$ can be used, but only if the CPU can tolerate the amount of advance provided by $\overline{SACK}$. If $\overline{SACK}$ arrives too early to provide the appropriate number of WAIT states, then either $\overline{XACK}$ or a delayed form of $\overline{SACK}$ should be used.

Write Cycles

Write cycles are similiar to Normal Read cycles, except for the $\overline{WE}$ output. $\overline{WE}$ is held inactive for Read cycles, but goes active for Write cycles. All 8202A Write cycles are "early-write" cycles; $\overline{WE}$ goes active before $\overline{CAS}$ goes active by an amount of time sufficient to keep the dynamic RAM output buffers turned off.

General System Considerations

All memory requests (Normal Reads, Advanced Reads, Writes) are qualified by the $\overline{PCS}$ input. $\overline{PCS}$ should be stable, either active or inactive, prior to the leading edge of $\overline{RD}$, $\overline{WR}$, or ALE. Systems which use battery backup should pullup $\overline{PCS}$ to prevent erroneous memory requests, and should also pullup $\overline{WR}$ to keep the 8202A out of its test mode.

In order to minimize propagation delay, the 8202A uses an inverting address multiplexer without latches. The system must provide adequate address setup and hold times to guarantee $\overline{RAS}$ and $\overline{CAS}$ setup and hold times for the RAM. The 8202A t_{AD} AC parameter should be used for this system calculation.

The B0-B1 inputs are similiar to the address inputs in that they are not latched. B0 and B1 should not be changed during a memory cycle, since they directly control which $\overline{RAS}$ output is activated.

The 8202A uses a two-stage synchronizer for the memory request inputs ($\overline{RD}$, $\overline{WR}$, ALE), and a separate two stage synchronizer for the external refresh input (REFRQ). As with any synchronizer, there is always a finite probability of metastable states inducing system errors. The 8202A synchronizer was designed to have a system error rate less than 1 memory cycle every three years based on the full operating range of the 8202A.

A microprocessor system is concerned with the time data is valid after $\overline{RD}$ goes low. See Figure 7. In order to calculate memory read access times, the dynamic RAM's A.C. specifications must be examined, especially the RAS-access time (t_{RAC}) and the CAS-access time (t_{CAC}). Most configurations will be CAS-access limited; i.e., the data from the RAM will be stable $t_{cc,max}$ (8202A) + t_{CAC} (RAM) after a memory read cycle is started. Be sure to add any delays (due to buffers, data latches, etc.) to calculate the overall read access time.

Since the 8202A normally performs "early-write" cycles, the data must be stable at the RAM data inputs by the time $\overline{CAS}$ goes active, including the RAM's data setup time. If the system does not normally guarantee sufficient write data setup, you must either delay the $\overline{WR}$ input signal or delay the 8202A $\overline{WE}$ output.

Delaying the $\overline{WR}$ input will delay all 8202A timing, including the READY handshake signals, $\overline{SACK}$ and $\overline{XACK}$, which may increase the number of WAIT states generated by the CPU.

If the $\overline{WE}$ output is externally delayed beyond the $\overline{CAS}$ active transition, then the RAM will use the falling edge of $\overline{WE}$ to strobe the write data into the RAM. This $\overline{WE}$ transition should not occur too late during the CAS active transition, or else the $\overline{WE}$ to $\overline{CAS}$ requirements of the RAM will not be met.

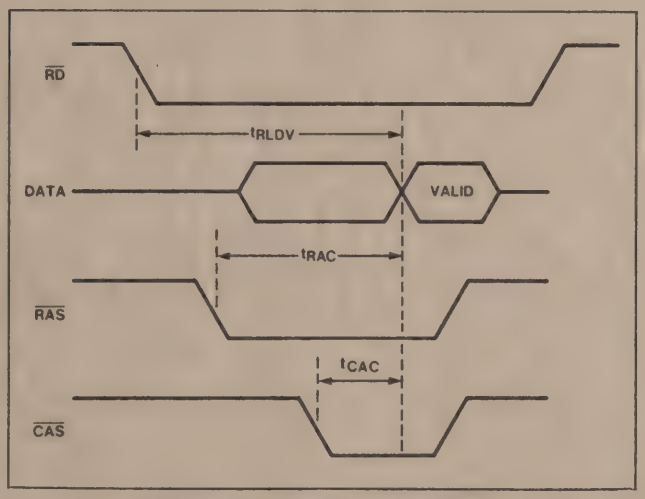


Figure 7. Read Access Time

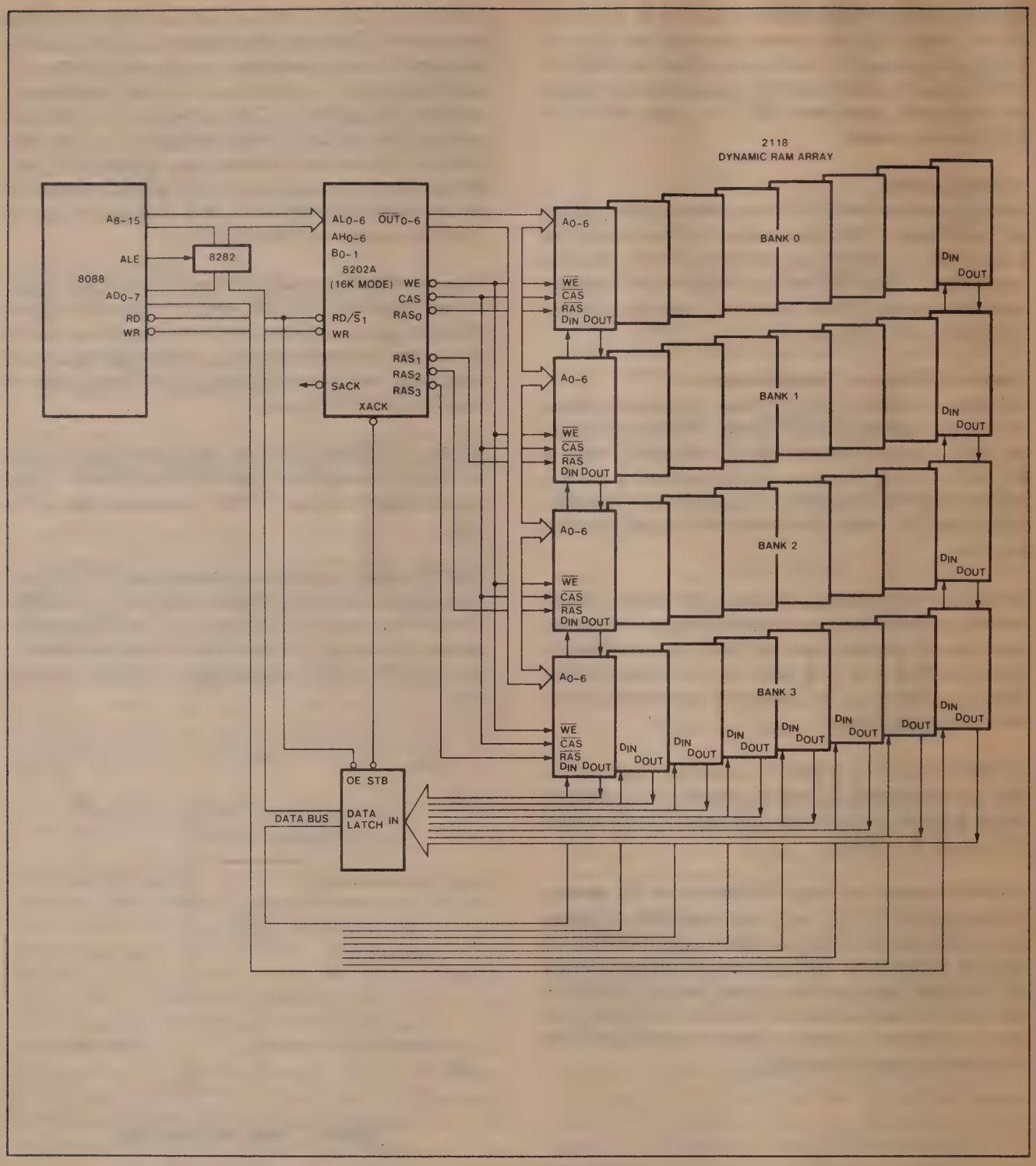


Figure 8. Typical 8088 System

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias .....0°C to 70°C
Storage Temperature .....−65°C to +150°C
Voltage On any Pin
With Respect to Ground .....−0.5V to +7V[4]
Power Dissipation .....1.5 Watts

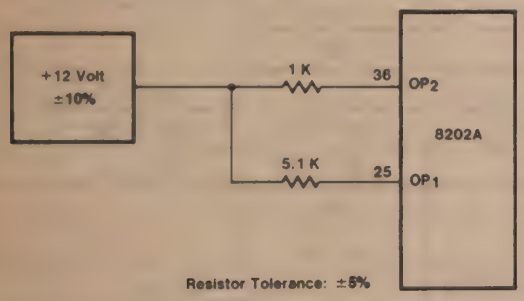
**NOTE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS

T_A = 0°C to 70°C; V_{CC} = 5.0V ± 10%, V_{CC} = 5.0V ± 5% for 8202A-3, GND = 0V

Symbol	Parameter	Min	Max	Units	Test Conditions
V_C	Input Clamp Voltage		−1.0	V	I_C = −5 mA
I_{CC}	Power Supply Current		270	mA	
I_F	Forward Input Current				
	CLK		−2.0	mA	V_F = 0.45V
	All Other Inputs[3]		−320	μA	V_F = 0.45V
I_R	Reverse Input Current[3]		40	μA	$V_R = V_{CC}$ (Note 1)
V_{OL}	Output Low Voltage				
	$\overline{SACK}$, $\overline{XACK}$		0.45	V	I_{OL} = 5 mA
	All Other Outputs		0.45	V	I_{OL} = 3 mA
V_{OH}	Output High Voltage				V_{IL} = 0.65V
	$\overline{SACK}$, $\overline{XACK}$	2.4		V	I_{OH} = −1 mA
	All Other Outputs	2.6		V	I_{OH} = −1 mA
V_{IL}	Input Low Voltage		0.8	V	V_{CC} = 5.0V (Note 2)
V_{IH1}	Input High Voltage	2.0		V	V_{CC} = 5.0V
V_{IH2}	Option Voltage			V	(Note 4)
C_{IN}	Input Capacitance		30	pF	F = 1 MHz V_{BIAS} = 2.5V, V_{CC} = 5V T_A = 25°C

NOTES:
1. I_R = 200μA for pin 37 (CLK) for external clock mode.
2. For test mode RD & WR must be held at GND.
3. Except for pin 36.
4. 8202A-1 and 8202A-3 supports both OP_1 and OP_2. 8202A only supports OP_2.



A.C. CHARACTERISTICS

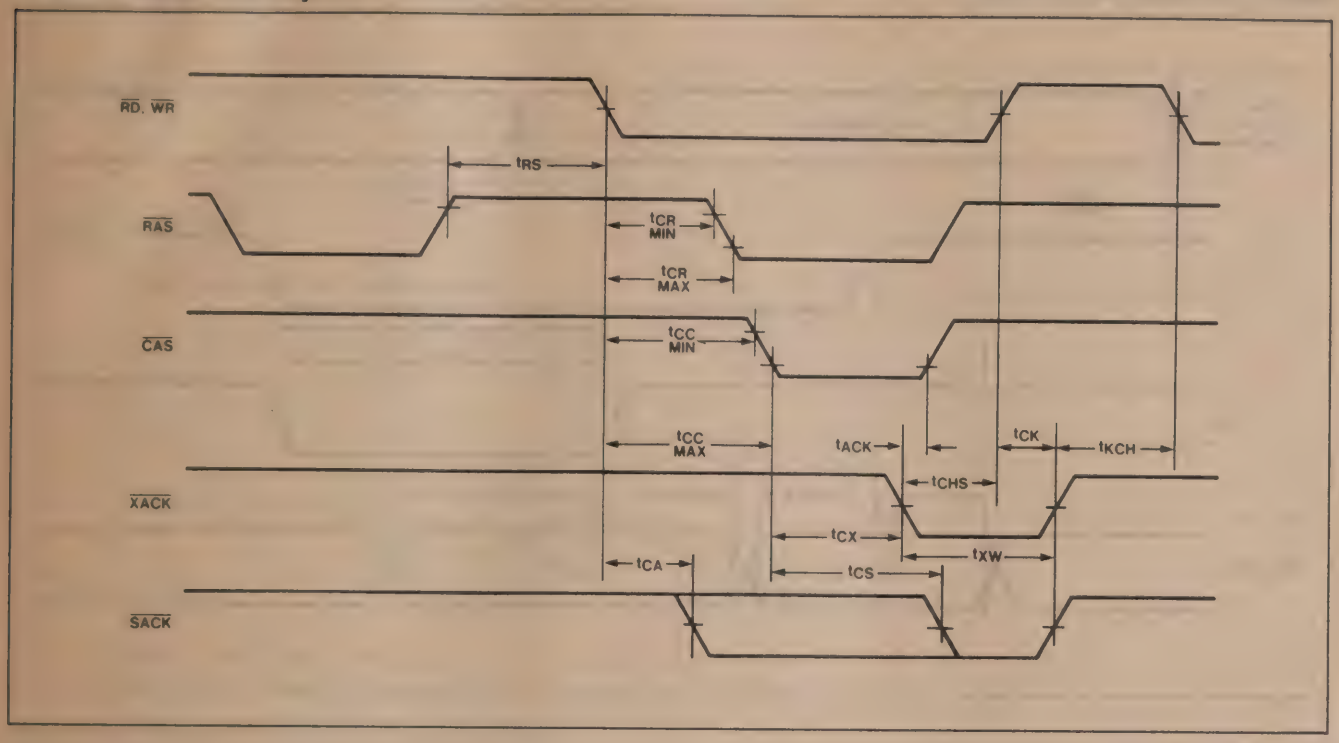
T_A = 0°C to 70°C, V_{CC} = 5V ± 10%, V_{CC} = 5V ± 5% for 8202A-3

Measurements made with respect to $\overline{RAS}_0$-$\overline{RAS}_3$, $\overline{CAS}$, $\overline{WE}$, $\overline{OUT}_0$-$\overline{OUT}_6$ are at 2.4V and 0.8V. All other pins are measured at 1.5V. All times are in nsec.

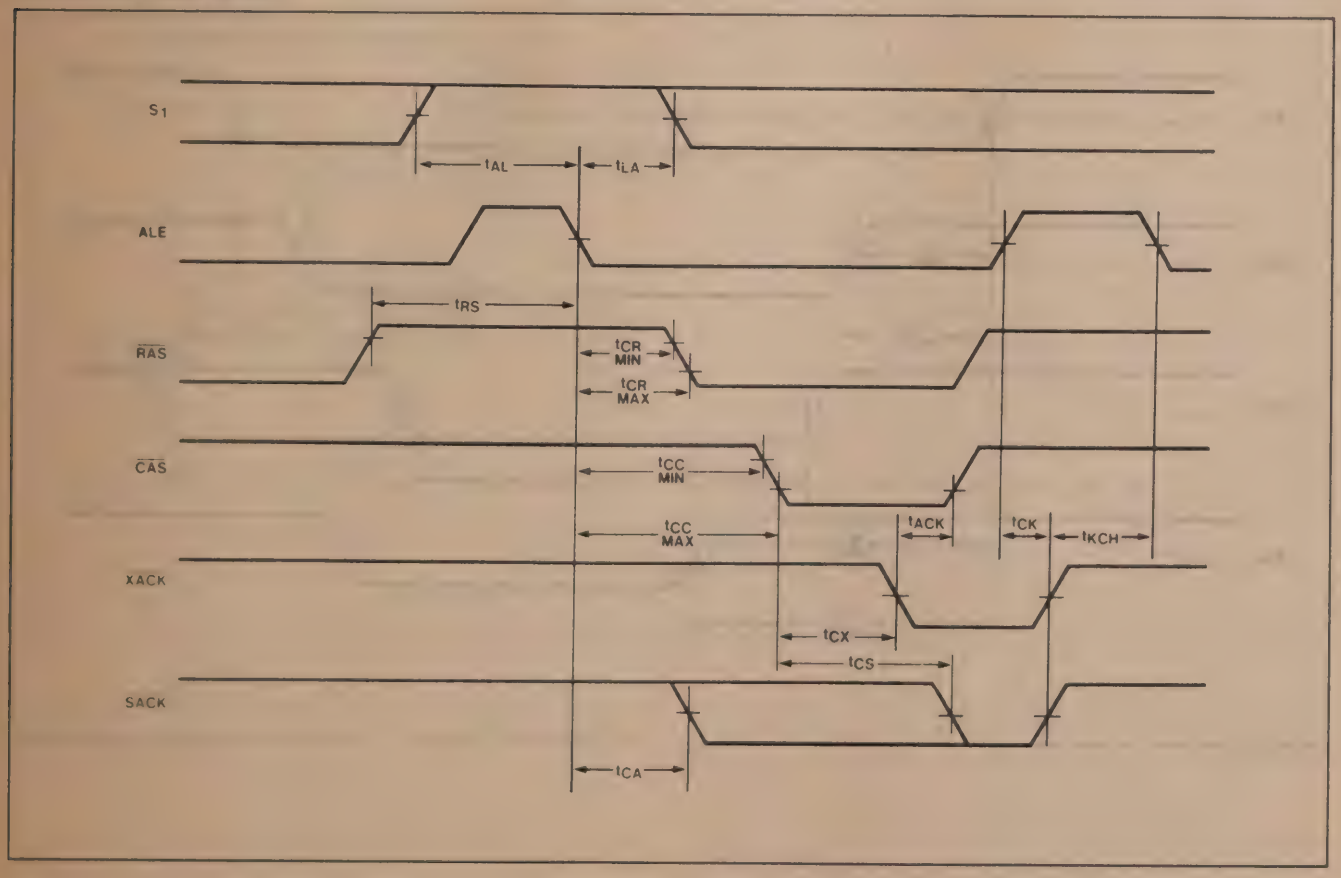
Symbol	Parameter	Min	Max	Notes
t_P	Clock Period	40	54	
t_{PH}	External Clock High Time	20		
t_{PL}	External Clock Low Time—above (>) 20 mHz	17		
t_{PL}	External Clock Low Time—below (<) 20 mHz	20		
t_{RC}	Memory Cycle Time	10tp − 30	12tp	4, 5
t_{REF}	Refresh Time (128 cycles—16K mode)	264tp	288tp	
t_{RP}	$\overline{RAS}$ Precharge Time	4tp − 30		
t_{RSH}	$\overline{RAS}$ Hold After $\overline{CAS}$	5tp − 30		3
t_{ASR}	Address Setup to $\overline{RAS}$	tp − 30		3
t_{RAH}	Address Hold From $\overline{RAS}$	tp − 10		3
t_{ASC}	Address Setup to $\overline{CAS}$	tp − 30		3
t_{CAH}	Address Hold from $\overline{CAS}$	5tp − 20		3
t_{CAS}	$\overline{CAS}$ Pulse Width	5tp − 10		
t_{WCS}	$\overline{WE}$ Setup to $\overline{CAS}$	tp − 40		
t_{WCH}	$\overline{WE}$ Hold After $\overline{CAS}$	5tp − 35		8
t_{RS}	$\overline{RD}$, $\overline{WR}$, ALE, REFRQ delay from $\overline{RAS}$	5tp		
t_{MRP}	$\overline{RD}$, $\overline{WR}$ setup to $\overline{RAS}$	0		5
t_{RMS}	REFRQ setup to $\overline{RD}$, $\overline{WR}$	2tp		
t_{RMP}	REFRQ setup to $\overline{RAS}$	2tp		5
t_{PCS}	$\overline{PCS}$ Setup to $\overline{RD}$, $\overline{WR}$, ALE	20		
t_{AL}	S1 Setup to ALE	15		
t_{LA}	S1 Hold from ALE	30		
t_{CR}	$\overline{RD}$, $\overline{WR}$, ALE to $\overline{RAS}$ Delay	tp + 30	2tp + 70	2
t_{CC}	$\overline{RD}$, $\overline{WR}$, ALE to $\overline{CAS}$ Delay	3tp + 25	4tp + 85	2
t_{SC}	CMD Setup to Clock	15		1
t_{MRS}	$\overline{RD}$, $\overline{WR}$ setup to REFRQ	5		
t_{CA}	$\overline{RD}$, $\overline{WR}$, ALE to $\overline{SACK}$ Delay		2tp + 47	2, 9
t_{CX}	$\overline{CAS}$ to $\overline{XACK}$ Delay	5tp − 25	5tp + 20	
t_{CS}	$\overline{CAS}$ to $\overline{SACK}$ Delay	5tp − 25	5tp + 40	2, 10
t_{ACK}	$\overline{XACK}$ to $\overline{CAS}$ Setup	10		
t_{XW}	$\overline{XACK}$ Pulse Width	tp − 25		7
t_{CK}	$\overline{SACK}$, $\overline{XACK}$ turn-off Delay		35	
t_{KCH}	CMD Inactive Hold after $\overline{SACK}$, $\overline{XACK}$	10		
t_{LL}	REFRQ Pulse Width	20		
t_{CHS}	CMD Hold Time	30		11
t_{RFR}	REFRQ to $\overline{RAS}$ Delay		4tp + 100	6
t_{WW}	$\overline{WR}$ to $\overline{WE}$ Delay	0	50	8
t_{AD}	CPU Address Delay	0	40	3

WAVEFORMS

Normal Read or Write Cycle

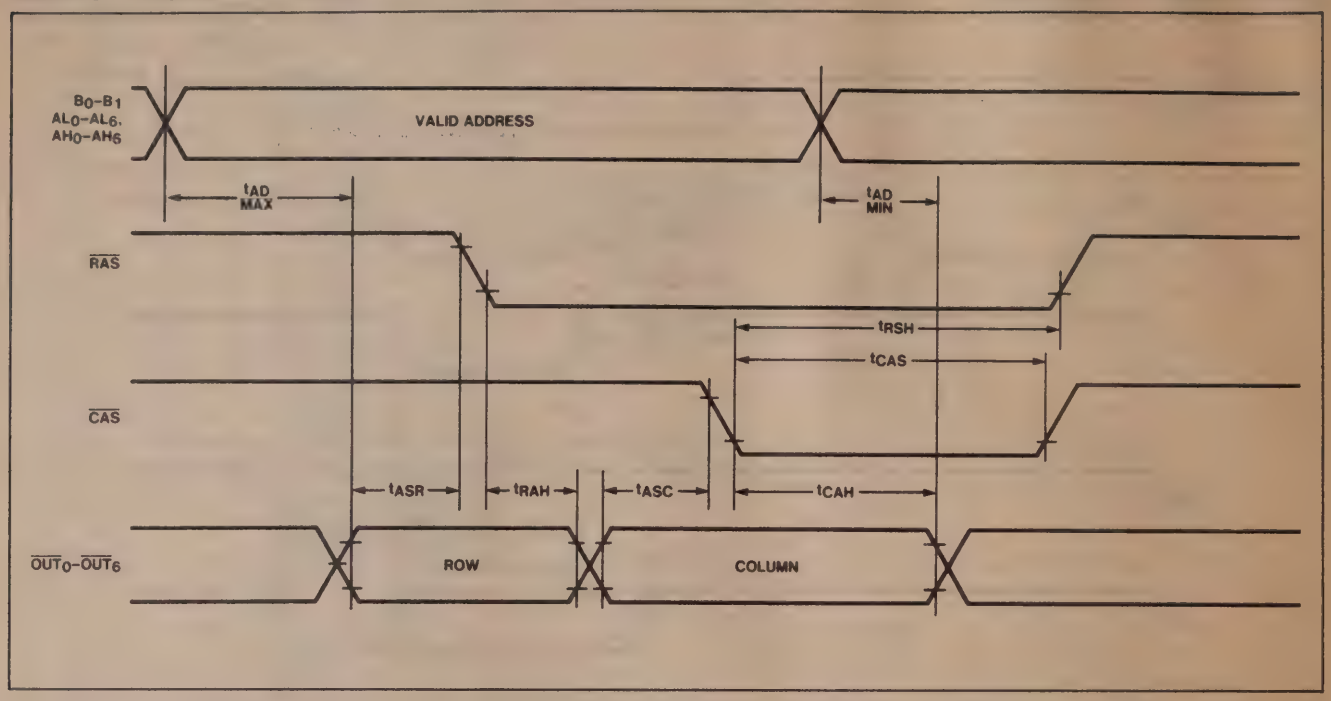


Advanced Read Mode

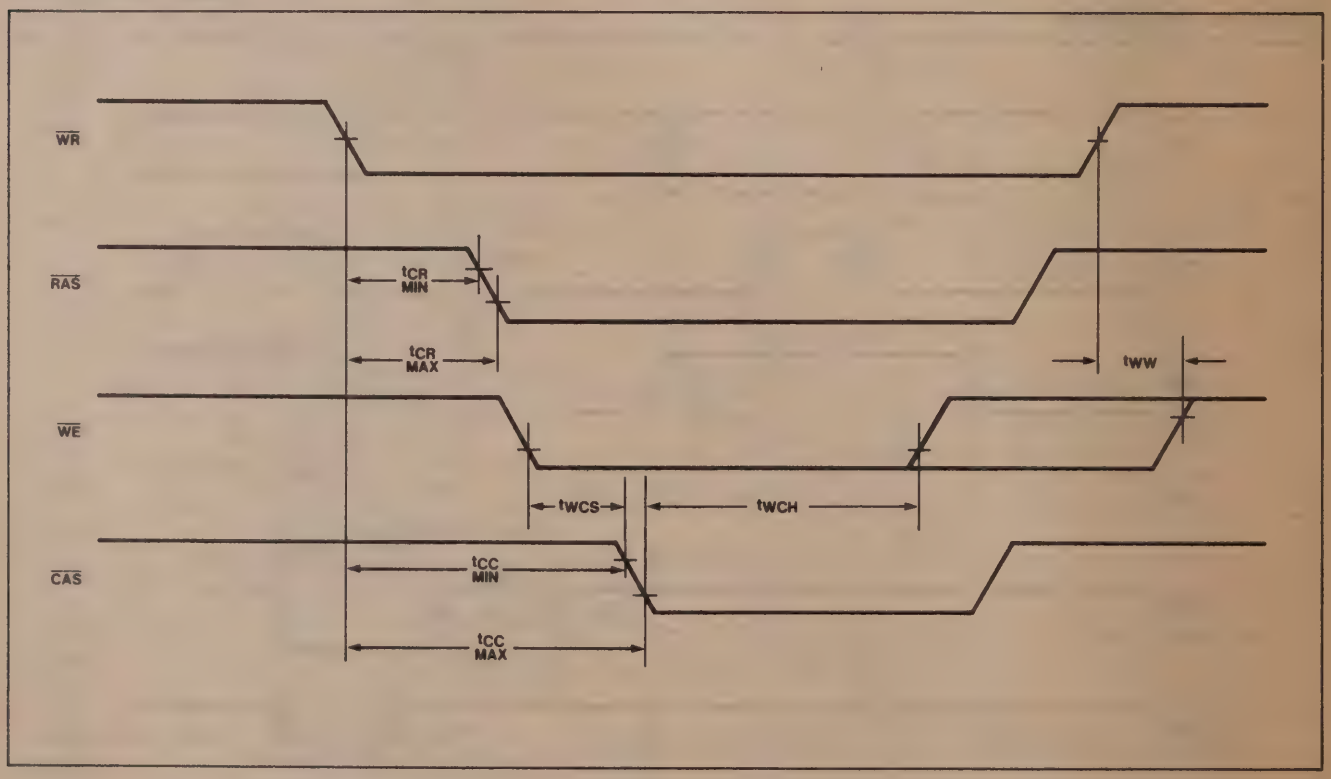


WAVEFORMS (cont'd)

Memory Compatibility Timing

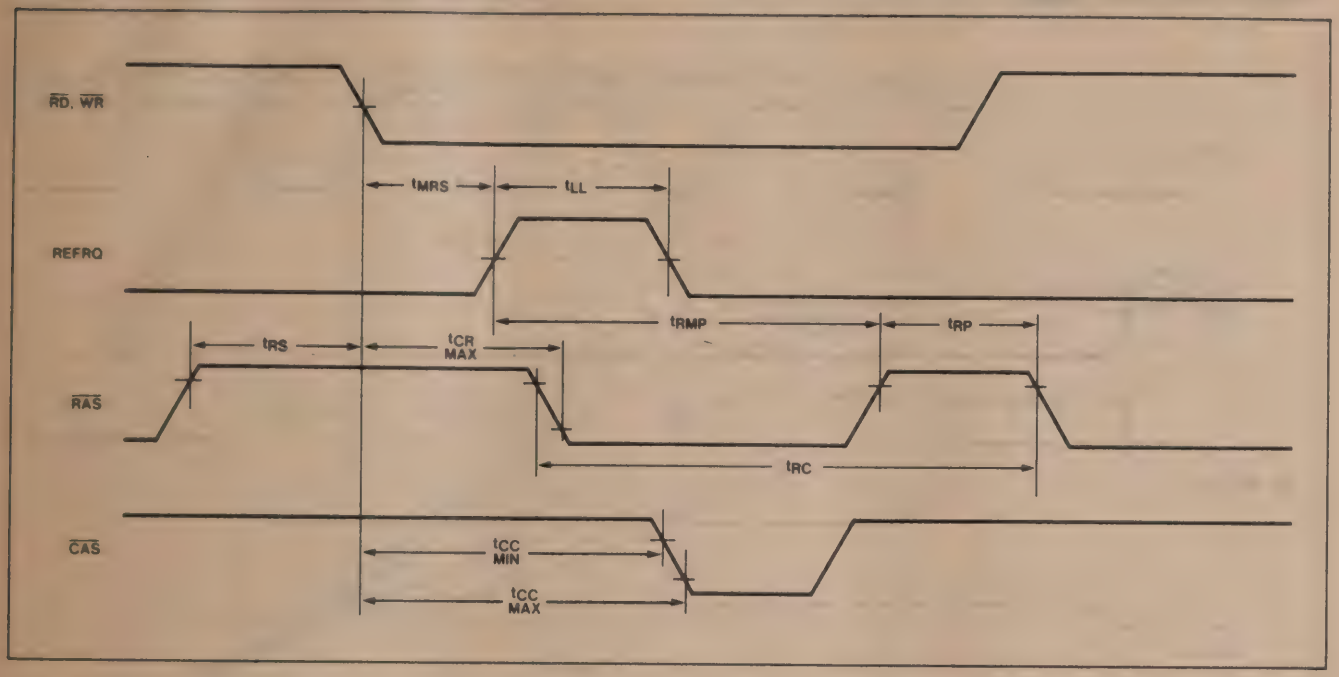


Write Cycle Timing

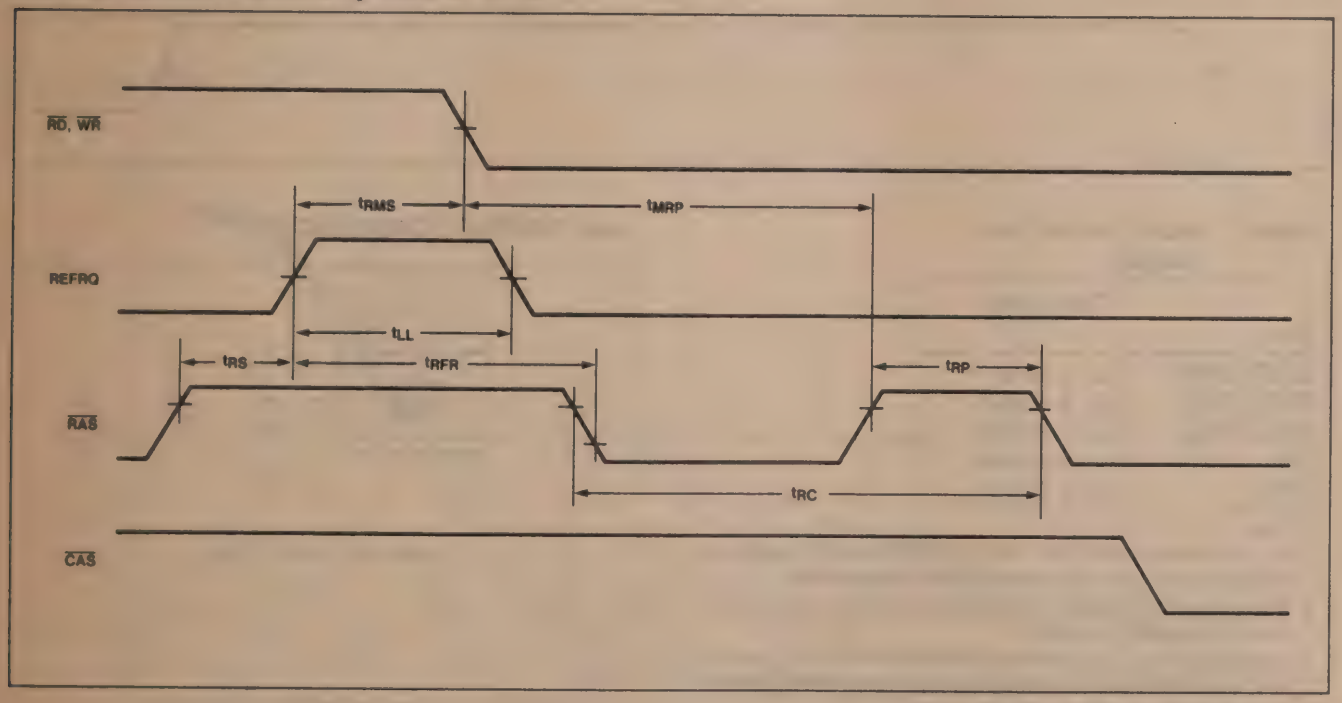


WAVEFORMS (cont'd)

Read or Write Followed By External Refresh



External Refresh Followed By Read or Write



WAVEFORMS (cont'd)

Clock And System Timing

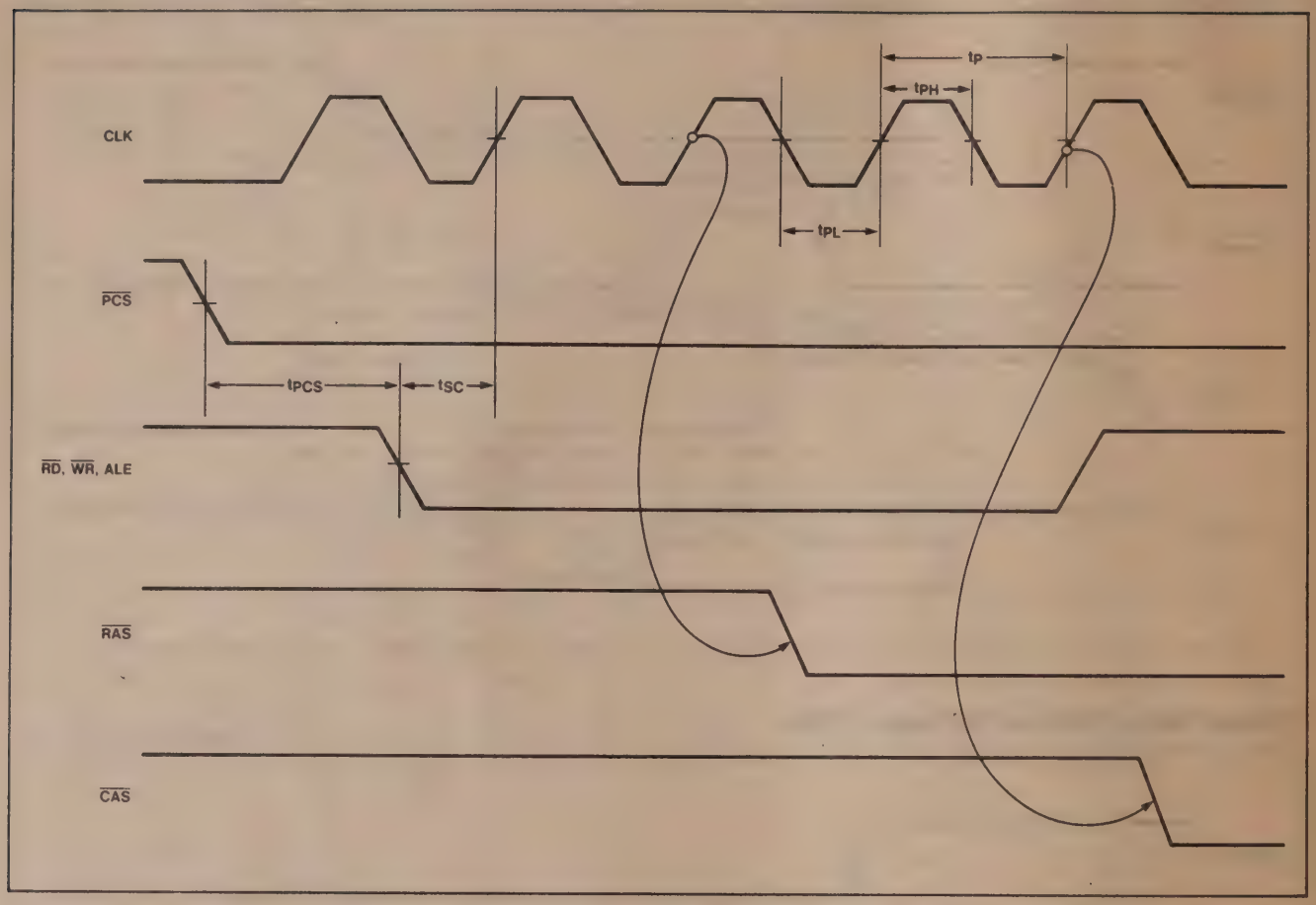


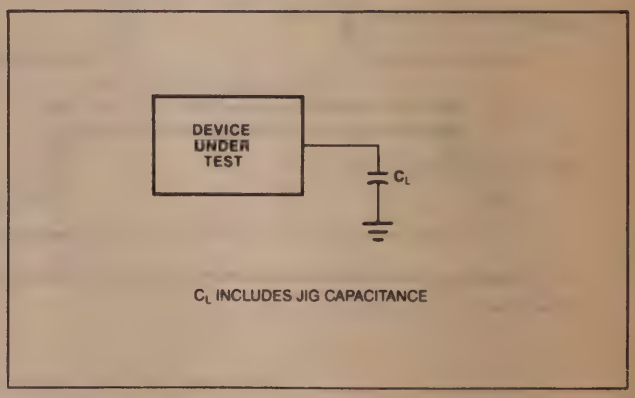
Table 2 8202A Output Test Loading.

Pin	Test Load
$\overline{SACK}$, $\overline{XACK}$	C_L = 30 pF
$\overline{OUT_0}$-$\overline{OUT_6}$	C_L = 160 pF
$\overline{RAS_0}$-$\overline{RAS_3}$	C_L = 60 pF
$\overline{WE}$	C_L = 224 pF
$\overline{CAS}$	C_L = 320 pF

NOTES:

1. t_{SC} is a reference point only. ALE, $\overline{RD}$, $\overline{WR}$, and REFRQ inputs do not have to be externally synchronized to 8202A clock.
2. If t_{RS} min and t_{MRS} min are met then, t_{CA}, t_{CR}, and t_{CC} are valid, otherwise t_{CS} is valid.
3. t_{ASR}, t_{RAH}, t_{ASC}, t_{CAH}, and t_{RSH} depend upon B0-B1 and CPU address remaining stable throughout the memory cycle. The address inputs are not latched by the 8202A.
4. For back-to-back refresh cycles, t_{RC} max = 13tp
5. t_{RC} max is valid only if t_{RMP} min is met (READ, WRITE followed by REFRESH) or t_{MRP} min is met (REFRESH followed by READ, WRITE).
6. t_{RFR} is valid only if t_{RS} min and t_{RMS} min are met.
7. t_{XW} min applies when $\overline{RD}$, $\overline{WR}$ has already gone high. Otherwise $\overline{XACK}$ follows $\overline{RD}$, $\overline{WR}$.
8. $\overline{WE}$ goes high according to t_{WCH} or t_{WW}, whichever occurs first.

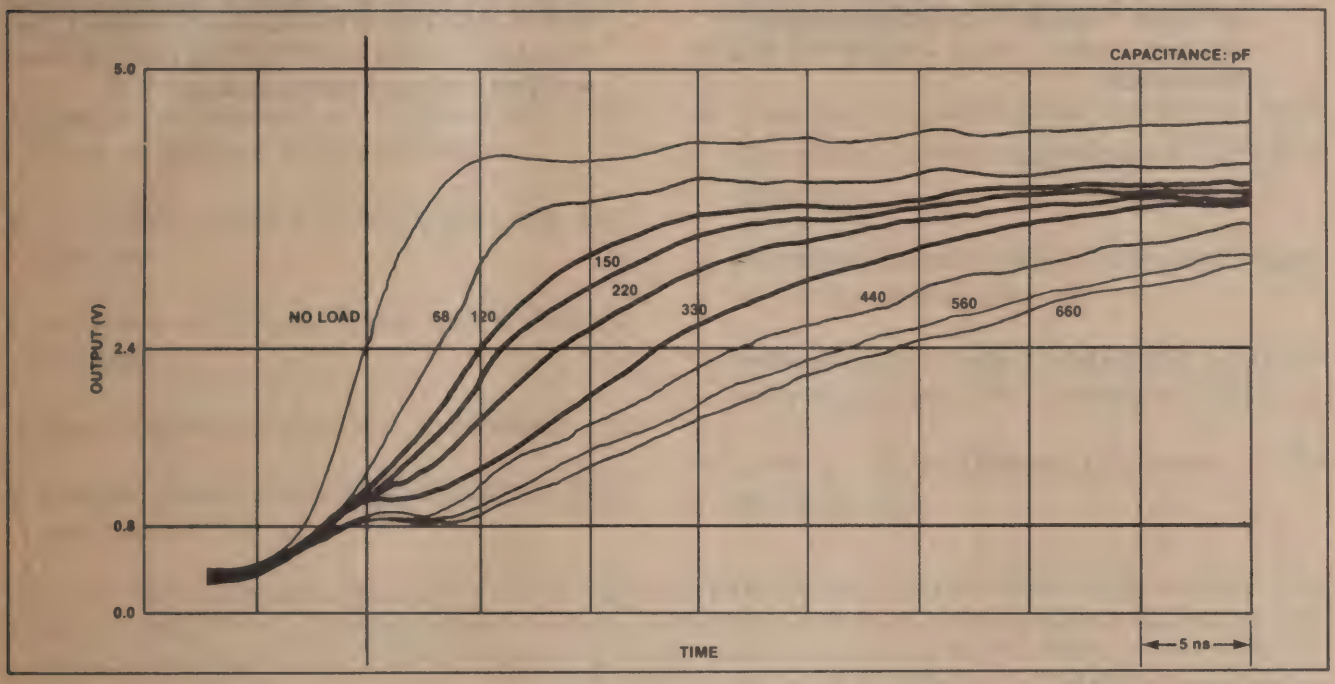
A.C. TESTING LOAD CIRCUIT

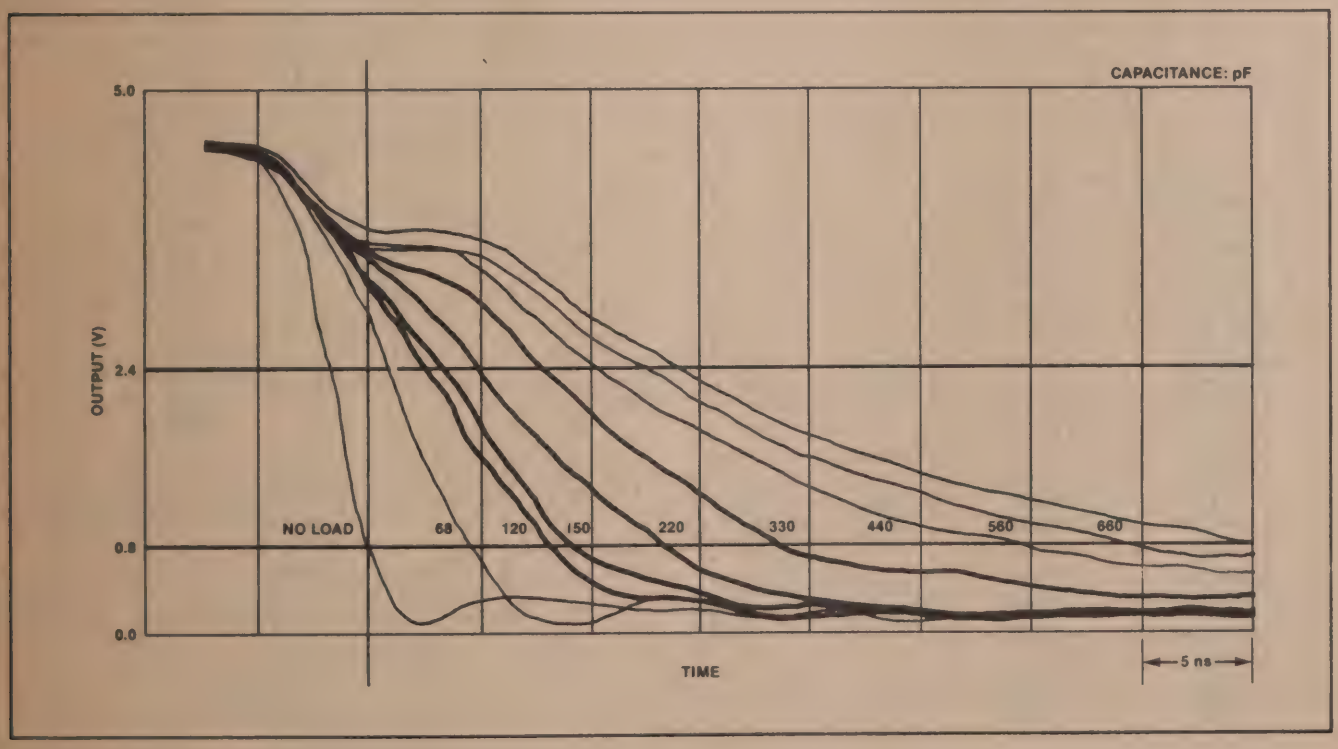


9. t_{CA} applies only when in normal $\overline{SACK}$ mode.
10. t_{CS} applies only when in delayed $\overline{SACK}$ mode.
11. t_{CHS} must be met only to ensure a $\overline{SACK}$ active pulse when in delayed $\overline{SACK}$ mode. $\overline{XACK}$ will always be activated for at least t_{XW} (t_P−25 nS). Violating t_{CHS} min does not otherwise affect device operation.

The typical rising and falling characteristic curves for the $\overline{OUT}$, $\overline{RAS}$, $\overline{CAS}$ and $\overline{WE}$ output buffers can be used to determine the effects of capacitive loading on the A.C. Timing Parameters. Using this design tool in conjunction with the timing waveforms, the designer can determine typical timing shifts based on system capacitive load.

A.C. CHARACTERISTICS FOR DIFFERENT CAPACITIVE LOADS





NOTE:
Use the Test Load as the base capacitance for estimating timing shifts for system critical timing parameters.

MEASUREMENT CONDITIONS:
$T_A = 25°C$
$V_{CC} = +5V$
$t_p = 50$ ns
Pins not measured are loaded with the Test Load capacitance.

205215-001

Example: Find the effect on t_{CR} and t_{CC} using 64 2118 Dynamic RAMs configured in 4 banks.

1. Determine the typical RAS and CAS capacitance: From the data sheet RAS = 4 pF and CAS = 4 pF.
   ∴ RAS load = 64 pF + board capacitance.
   CAS load = 256 pF + board capacitance.
   Assume 2 pF/in (trace length) for board capacitance.
2. From the waveform diagrams, we determine that the falling edge timing is needed for t_{CR} and t_{CC}. Next find the curve that *best* approximates the test load; i.e., 68 pF for RAS and 330 pF for CAS.
3. If we use 72 pF for RAS loading, then the t_{CR} (max.) spec should be increased by *about* 1 ns. Similarly if we use 288 pF for CAS, then t_{CC} (min.) and (max.) should decrease about 1 ns.

intel

8203
64K DYNAMIC RAM CONTROLLER

- **Provides All Signals Necessary to Control 64K (2164) and 16K (2117, 2118) Dynamic Memories**
- **Directly Addresses and Drives Up to 64 Devices Without External Drivers**
- **Provides Address Multiplexing and Strobes**
- **Provides a Refresh Timer and a Refresh Counter**
- **Provides Refresh/Access Arbitration**
- **Internal Clock Capability with the 8203-1 and the 8203-3**
- **Fully Compatible with Intel® 8080A, 8085A, iAPX 88, and iAPX 86 Family Microprocessors**
- **Decodes CPU Status for Advanced Read Capability in 16K mode with the 8203-1 and the 8203-3.**
- **Provides System Acknowledge and Transfer Acknowledge Signals**
- **Refresh Cycles May be Internally or Externally Requested (For Transparent Refresh)**
- **Internal Series Damping Resistors on All RAM Outputs**

The Intel® 8203 is a Dynamic Ram System Controller designed to provide all signals necessary to use 2164, 2118 or 2117 Dynamic RAMs in microcomputer systems. The 8203 provides multiplexed addresses and address strobes, refresh logic, refresh/access arbitration. Refresh cycles can be started internally or externally. The 8203-1 and the 8203-3 support an internal crystal oscillator and Advanced Read Capability. The 8203-3 is a ±5% V_{CC} part.

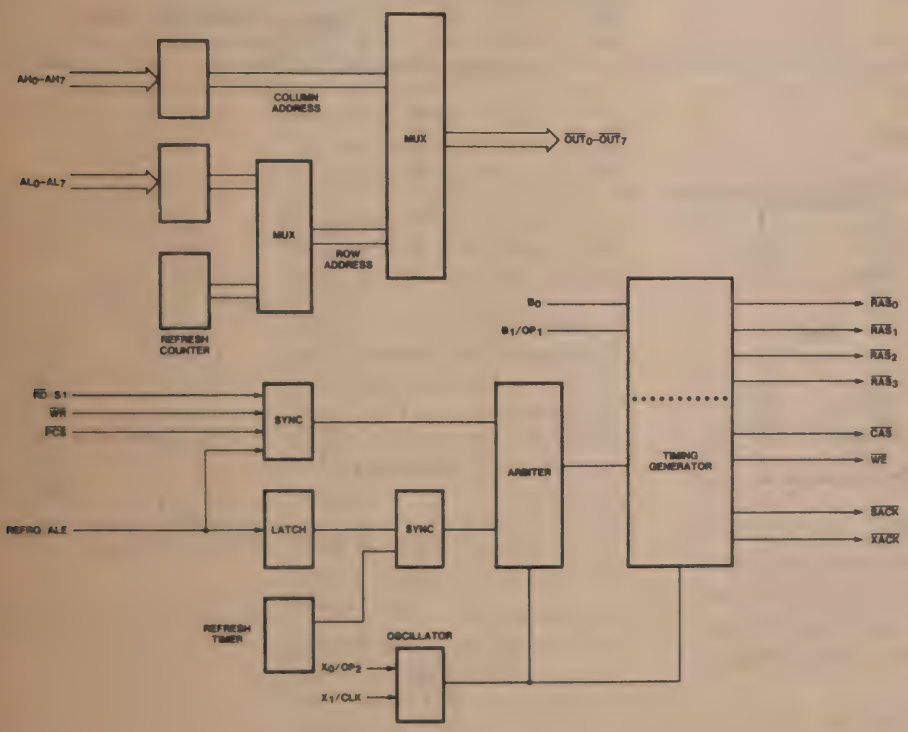


Figure 1. 8203 Block Diagram

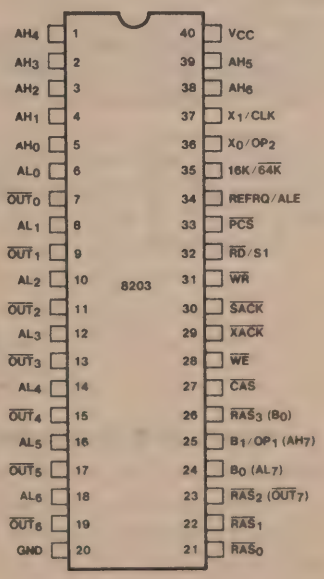


Figure 2. Pin Configuration

Intel Corporation Assumes No Responsibility for the Use of Any Circuitry Other Than Circuitry Embodied in an Intel Product. No Other Circuit Patent Licenses are Implied.
©INTEL CORPORATION, 1982

JANUARY 1985
ORDER NUMBER: 210444-004

ABSOLUTE MAXIMUM RATINGS*

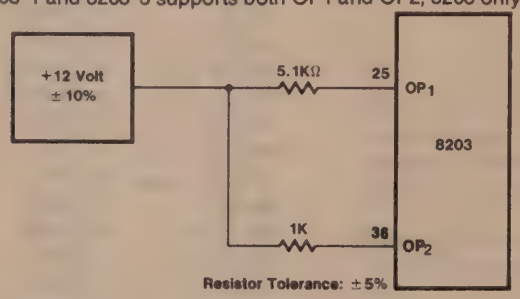
Ambient Temperature Under Bias .....0°C to 70°C
Storage Temperature .....−65°C to +150°C
Voltage On any Pin
With Respect to Ground .....−0.5V to +7V[4]
Power Dissipation .....1.6 Watts

**NOTE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS T_A = 0°C to 70°C; V_{CC} = 5.0V ± 10% (5.0V ± 5% for 8203-3); GND = 0V

Symbol	Parameter	Min	Max	Units	Test Conditions
V_C	Input Clamp Voltage		−1.0	V	I_C = −5 mA
I_{CC}	Power Supply Current		290	mA	
I_F	Forward Input Current CLK, 64K/16K Mode select All Other Inputs[3]		 −2.0 −320	 mA μA	 V_F = 0.45V V_F = 0.45V
I_R	Reverse Input Current[3]		40	μA	V_R = V_{CC}; Note 1, 5
V_{OL}	Output Low Voltage $\overline{SACK}$, $\overline{XACK}$ All Other Outputs		 0.45 0.45	 V V	 I_{OL} = 5 mA I_{OL} = 3 mA
V_{OH}	Output High Voltage $\overline{SACK}$, $\overline{XACK}$ All Other Outputs	 2.4 2.6		 V V	V_{IL} = 0.65 V I_{OH} = −1 mA I_{OH} = −1 mA
V_{IL}	Input Low Voltage		0.8	V	V_{CC} = 5.0V (Note 2)
V_{IH1}	Input High Voltage	2.0	V_{CC}	V	V_{CC} = 5.0V
V_{IH2}	Option Voltage		V_{CC}	V	(Note 4)
C_{IN}	Input Capacitance		30	pF	F = 1 MHz V_{BIAS} = 2.5V, V_{CC} = 5V T_A = 25°C

NOTES:
1. I_R = 200 μA for pin 37 (CLK).
2. For test mode $\overline{RD}$ & $\overline{WR}$ must be held at GND.
3. Except for pin 36 in XTAL mode.
4. 8203-1 and 8203-3 supports both OP1 and OP2, 8203 only supports OP2.



5. I_R = 150μA for pin 35 (Mode Select 16K/64K)

Table 1. Pin Descriptions

Symbol	Pin No.	Type	Name and Function
AL_0 AL_1 AL_2 AL_3 AL_4 AL_5 AL_6	6 8 10 12 14 16 18	I I I I I I I	**Address Low:** CPU address inputs used to generate memory row address.
AH_0 AH_1 AH_2 AH_3 AH_4 AH_5 AH_6	5 4 3 2 1 39 38	I I I I I I I	**Address High:** CPU address inputs used to generate memory column address.
B_0/AL_7 $B_1/OP_1/AH_7$	24 25	I I	**Bank Select Inputs:** Used to gate the appropriate RAS output for a memory cycle. B_1/OP_1 option used to select the Advanced Read Mode. (Not available in 64K mode.) See Figure 5. When in 64K RAM Mode, pins 24 and 25 operate as the AL_7 and AH_7 address inputs.
$\overline{PCS}$	33	I	**Protected Chip Select:** Used to enable the memory read and write inputs. Once a cycle is started, it will not abort even if $\overline{PCS}$ goes inactive before cycle completion.
$\overline{WR}$	31	I	**Memory Write Request.**
$\overline{RD}$/S1	32	I	**Memory Read Request:** S1 function used in Advanced Read mode selected by OP_1 (pin 25).
REFRQ/ALE	34	I	**External Refresh Request:** ALE function used in Advanced Read mode, selected by OP_1 (pin 25).
$\overline{OUT}_0$ $\overline{OUT}_1$ $\overline{OUT}_2$ $\overline{OUT}_3$ $\overline{OUT}_4$ $\overline{OUT}_5$ $\overline{OUT}_6$	7 9 11 13 15 17 19	O O O O O O O	**Output of the Multiplexer:** These outputs are designed to drive the addresses of the Dynamic RAM array. (Note that the $\overline{OUT}_{0\text{-}7}$ pins do not require inverters or drivers for proper operation.)
$\overline{WE}$	28	O	**Write Enable:** Drives the Write Enable inputs of the Dynamic RAM array.
$\overline{CAS}$	27	O	**Column Address Strobe:** This output is used to latch the Column Address into the Dynamic RAM array.
$\overline{RAS}_0$ $\overline{RAS}_1$ $\overline{RAS}_2/\overline{OUT}_7$ $\overline{RAS}_3/B_0$	21 22 23 26	O O O I/O	**Row Address Strobe:** Used to latch the Row Address into the bank of dynamic RAMs, selected by the 8203 Bank Select pins (B_0, B_1/OP_1). In 64K mode, only $\overline{RAS}_0$ and $\overline{RAS}_1$ are available; pin 23 operates as $\overline{OUT}_7$ and pin 26 operates as the B_0 bank select input.
$\overline{XACK}$	29	O	**Transfer Acknowledge:** This output is a strobe indicating valid data during a read cycle or data written during a write cycle. $\overline{XACK}$ can be used to latch valid data from the RAM array.
$\overline{SACK}$	30	O	**System Acknowledge:** This output indicates the beginning of a memory access cycle. It can be used as an advanced transfer acknowledge to eliminate wait states. (Note: If a memory access request is made during a refresh cycle, $\overline{SACK}$ is delayed until $\overline{XACK}$ in the memory access cycle).
X_0/OP_2 X_1/CLK	36 37	I/O I/O	**Oscillator Inputs:** These inputs are designed for a quartz crystal to control the frequency of the oscillator. If X_0/OP_2 is shorted to pin 40 (V_{CC}) or if X_0/OP_2 is connected to +12V through a 1KΩ resistor then X_1/CLK becomes a TTL input for an external clock. (Note: Crystal mode for the 8203-1 and the 8203-3 only).
16K/$\overline{64K}$	35	I	**Mode Select:** This input selects 16K mode (2117, 2118) or 64K mode (2164). Pins 23-26 change function based on the mode of operation.
V_{CC}	40		**Power Supply:** +5V.
GND	20		**Ground.**

Functional Description

The 8203 provides a complete dynamic RAM controller for microprocessor systems as well as expansion memory boards. All of the necessary control signals are provided for 2164, 2118 and 2117 dynamic RAMs.

The 8203 has two modes, one for 16K dynamic RAMs and one for 64Ks, controlled by pin 35.

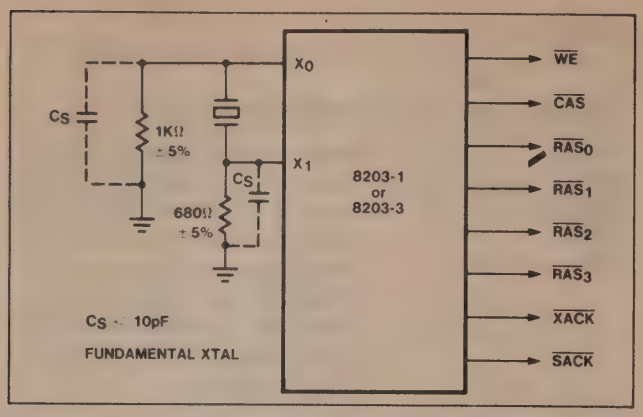


Figure 3. Crystal Operation for the 8203-1 and 8203-3

All 8203 timing is generated from a single reference clock. This clock is provided via an external oscillator or an on-chip crystal oscillator. All output signal transitions are synchronous with respect to this clock reference, except for the trailing edges of the CPU handshake signals $\overline{SACK}$ and $\overline{XACK}$.

CPU memory requests normally use the $\overline{RD}$ and $\overline{WR}$ inputs. The Advanced-Read mode allows ALE and S1 to be used in place of the $\overline{RD}$ input.

Failsafe refresh is provided via an internal timer which generates refresh requests. Refresh requests can also be generated via the REFRQ input.

An on-chip synchronizer/arbiter prevents memory and refresh requests from affecting a cycle in progress. The READ, WRITE, and external REFRESH requests may be asynchronous to the 8203 clock; on-chip logic will synchronize the requests, and the arbiter will decide if the requests should be delayed, pending completion of a cycle in progress.

16K/$\overline{64K}$ Option Selection

Pin 35 is a strap input that controls the two 8203 modes. Figure 4 shows the four pins that are multiplexed. In 16K mode (pin 35 tied to V_{CC} or left open), the 8203 has two Bank Select inputs to select one of four $\overline{RAS}$ outputs. In this mode, the 8203 is exactly compatible with the Intel 8202A Dynamic RAM Controller. In 64K mode (pin 35 tied to GND), there is only one Bank Select input (pin 26) to select the two $\overline{RAS}$ outputs. More than two banks of 64K dynamic RAM's can be used with external logic.

Other Option Selections

The 8203 has two strapping options. When OP_1 is selected (16K mode only), pin 32 changes from a $\overline{RD}$ input to an S1 input, and pin 34 changes from a REFRQ input to an ALE input. See "Refresh Cycles" and "Read Cycles" for more detail. OP_1 is selected by tying pin 25 to +12V through a 5.1K ohm resistor on the 8203-1 or 8203-3 only.

When OP_2 is selected, the internal oscillator is disabled and pin 37 changes from a crystal input (X_1) to a CLK input for an external TTL clock. OP_2 is selected by shorting pin 36 (X_0/OP_2) directly to pin 40 (V_{CC}). No current limiting resistor should be used. OP_2 may also be selected by tying pin 36 to +12V through a 1KΩ resistor.

Refresh Timer

The refresh timer is used to monitor the time since the last refresh cycle occurred. When the appropriate amount of time has elapsed, the refresh timer will request a refresh cycle. External refresh requests will reset the refresh timer.

Refresh Counter

The refresh counter is used to sequentially refresh all of the memory's rows. The 8-bit counter is incremented after every refresh cycle.

Pin #	16K Function	64K Function
23	$\overline{RAS_2}$	Address Output ($\overline{OUT_7}$)
24	Bank Select (B_0)	Address Input (AL_7)
25	Bank Select (B_1)	Address Input (AH_7)
26	$\overline{RAS_3}$	Bank Select (B_0)

Figure 4. 16K/$\overline{64K}$ Mode Selection

	Inputs		Outputs			
	B_1	B_0	$\overline{RAS_0}$	$\overline{RAS_1}$	$\overline{RAS_2}$	$\overline{RAS_3}$
16K Mode	0	0	0	1	1	1
	0	1	1	0	1	1
	1	0	1	1	0	1
	1	1	1	1	1	0
64K Mode	—	0	0	1	—	—
	—	1	1	0	—	—

Figure 5. Bank Selection

Description	Pin #	Normal Function	Option Function
$B1/OP1$ (16K only)/AH_7	25	Bank (RAS) Select	Advanced-Read Mode (8203-1, -3)
X_0/OP_2	36	Crystal Oscillator (8203-1 and 8203-3)	External Oscillator

Figure 6. 8203 Option Selection

Address Multiplexer

The address multiplexer takes the address inputs and the refresh counter outputs, and gates them onto the address outputs at the appropriate time. The address outputs, in conjunction with the $\overline{RAS}$ and $\overline{CAS}$ outputs, determine the address used by the dynamic RAMs for read, write, and refresh cycles. During the first part of a read or write cycle, AL_0–AL_7 are gated to $\overline{OUT}_0$–$\overline{OUT}_7$, then AH_0–AH_7 are gated to the address outputs.

During a refresh cycle, the refresh counter is gated onto the address outputs. All refresh cycles are RAS-only refresh ($\overline{CAS}$ inactive, $\overline{RAS}$ active).

To minimize buffer delay, the information on the address outputs is inverted from that on the address inputs.

$\overline{OUT}_0$–$\overline{OUT}_7$ do not need inverters or buffers unless additional drive is required.

Synchronizer/Arbiter

The 8203 has three inputs, REFRQ/ALE (pin 34), $\overline{RD}$ (pin 32) and $\overline{WR}$ (pin 31). The $\overline{RD}$ and $\overline{WR}$ inputs allow an external CPU to request a memory read or write cycle, respectively. The REFRQ/ALE input allows refresh requests to be requested external to the 8203.

All three of these inputs may be asynchronous with respect to the 8203's clock. The arbiter will resolve conflicts between refresh and memory requests, for both pending cycles and cycles in progress. Read and write requests will be given priority over refresh requests.

System Operation

The 8203 is always in one of the following states:

a) IDLE
b) TEST Cycle
c) REFRESH Cycle
d) READ Cycle
e) WRITE Cycle

The 8203 is normally in the IDLE state. Whenever one of the other cycles is requested, the 8203 will leave the IDLE state to perform the desired cycle. If no other cycles are pending, the 8203 will return to the IDLE state.

Test Cycle

The TEST Cycle is used to check operation of several 8203 internal functions. TEST cycles are requested by activating the $\overline{PCS}$, $\overline{RD}$ and $\overline{WR}$ inputs. The TEST Cycle will reset the refresh address counter and perform a WRITE Cycle. The TEST Cycle should not be used in normal system operation, since it would affect the dynamic RAM refresh.

Refresh Cycles

The 8203 has two ways of providing dynamic RAM refresh:

1) Internal (failsafe) refresh
2) External (hidden) refresh

Both types of 8203 refresh cycles activate all of the $\overline{RAS}$ outputs, while $\overline{CAS}$, $\overline{WE}$, $\overline{SACK}$, and $\overline{XACK}$ remain inactive.

Internal refresh is generated by the on-chip refresh timer. The timer uses the 8203 clock to ensure that refresh of all rows of the dynamic RAM occurs every 2 milliseconds (128 cycles) or every 4 milliseconds (256 cycles). If REFRQ is inactive, the refresh timer will request a refresh cycle every 10-16 microseconds.

External refresh is requested via the REFRQ input (pin 34). External refresh control is not available when the Advanced-Read mode is selected. External refresh requests are latched, then synchronized to the 8203 clock.

The arbiter will allow the refresh request to start a refresh cycle only if the 8203 is not in the middle of a cycle.

When the 8203 is in the idle state a simultaneous memory request and external refresh request will result in the memory request being honored first. This 8203 characteristic can be used to "hide" refresh cycles during system operation. A circuit similar to Figure 7 can be used to decode the CPU's instruction fetch status to generate an external refresh request. The refresh request is latched while the 8203 performs the instruction fetch; the refresh cycle will start immediately after the memory cycle is completed, even if the $\overline{RD}$ input has not gone inactive. If the CPU's instruction decode time is long enough, the 8203 can complete the refresh cycle before the next memory request is generated.

If the 8203 is not in the idle state then a simultaneous memory request and an external refresh request may result in the refresh request being honored first.

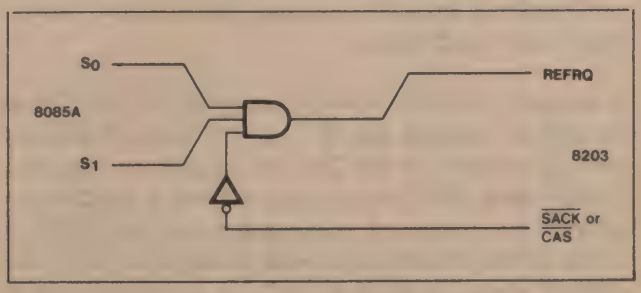


Figure 7. Hidden Refresh

Certain system configurations require complete external refresh requests. If external refresh is requested faster than the minimum internal refresh timer (t_{REF}), then, in effect, all refresh cycles will be caused by the external refresh request, and the internal refresh timer will never generate a refresh request.

Read Cycles

The 8203 can accept two different types of memory Read requests:

1) Normal Read, via the $\overline{RD}$ input
2) Advanced Read, using the S1 and ALE inputs (16K mode only)

The user can select the desired Read request configuration via the B1/OP1 hardware strapping option on pin 25.

	Normal Read	Advanced Read
Pin 25	B1 input	OP_1 (+12V)
Pin 32	RD input	S1 input
Pin 34	REFRQ input	ALE input
# RAM banks	4 ($\overline{RAS}_{0-3}$)	2 ($\overline{RAS}_{2-3}$)
Ext. Refresh	Yes	No

Figure 8. 8203 Read Options

Normal Reads are requested by activating the $\overline{RD}$ input, and keeping it active until the 8203 responds with an $\overline{XACK}$ pulse. The $\overline{RD}$ input can go inactive as soon as the command hold time (t_{CHS}) is met.

Advanced Read cycles are requested by pulsing ALE while S1 is active; if S1 is inactive (low) ALE is ignored. Advanced Read timing is similiar to Normal Read timing, except the falling edge of ALE is used as the cycle start reference.

If a Read cycle is requested while a refresh cycle is in progress, then the 8203 will set the internal delayed-SACK latch. When the Read cycle is eventually started, the 8203 will delay the active $\overline{SACK}$ transition until $\overline{XACK}$ goes active, as shown in the AC timing diagrams. This delay was designed to compensate for the CPU's READY setup and hold times. The delayed–SACK latch is cleared after every READ cycle.

Based on system requirements, either $\overline{SACK}$ or $\overline{XACK}$ can be used to generate the CPU READY signal. $\overline{XACK}$ will normally be used; if the CPU can tolerate an advanced READY, then $\overline{SACK}$ can be used, but only if the CPU can tolerate the amount of advance provided by $\overline{SACK}$. If $\overline{SACK}$ arrives too early to provide the appropriate number of WAIT states, then either $\overline{XACK}$ or a delayed form of $\overline{SACK}$ should be used.

Write Cycles

Write cycles are similiar to Normal Read cycles, except for the $\overline{WE}$ output. $\overline{WE}$ is held inactive for Read cycles, but goes active for Write cycles. All 8203 Write cycles are "early-write" cycles; $\overline{WE}$ goes active before $\overline{CAS}$ goes active by an amount of time sufficient to keep the dynamic RAM output buffers turned off.

General System Considerations

All memory requests (Normal Reads, Advanced Reads, Writes) are qualified by the $\overline{PCS}$ input. $\overline{PCS}$ should be stable, either active or inactive, prior to the leading edge of $\overline{RD}$, $\overline{WR}$, or ALE. Systems which use battery backup should pullup $\overline{PCS}$ to prevent erroneous memory requests.

In order to minimize propagation delay, the 8203 uses an inverting address multiplexer without latches. The system must provide adequate address setup and hold times to guarantee $\overline{RAS}$ and $\overline{CAS}$ setup and hold times for the RAM. The t_{AD} AC parameter should be used for this system calculation.

The B_0-B_1 inputs are similiar to the address inputs in that they are not latched. B_0 and B_1 should not be changed during a memory cycle, since they directly control which $\overline{RAS}$ output is activated.

The 8203 uses a two-stage synchronizer for the memory request inputs ($\overline{RD}$, $\overline{WR}$, ALE), and a separate two stage synchronizer for the external refresh input (REFRQ). As with any synchronizer, there is always a finite probability of metastable states inducing system errors. The 8203 synchronizer was designed to have a system error rate less than 1 memory cycle every three years based on the full operating range of the 8203.

A microprocessor system is concerned when the data is valid after $\overline{RD}$ goes low. See Figure 9. In order to calculate memory read access times, the dynamic RAM's A.C. specifications must be examined, especially the RAS-access time (t_{RAC}) and the CAS-access time (t_{CAC}). Most configurations will be CAS-access limited; i.e., the data from the RAM will be stable $t_{cc,max}$ (8203) + t_{CAC} (RAM) after a memory read cycle is started. Be sure to add any delays (due to buffers, data latches, etc.) to calculate the overall read access time.

Since the 8203 normally performs "early-write" cycles, the data must be stable at the RAM data inputs by the time $\overline{CAS}$ goes active, including the RAM's data setup time. If the system does not normally guarantee sufficient write data setup, you must either delay the $\overline{WR}$ input signal or delay the 8203 $\overline{WE}$ output.

Delaying the $\overline{WR}$ input will delay all 8203 timing, including the READY handshake signals, $\overline{SACK}$ and $\overline{XACK}$, which

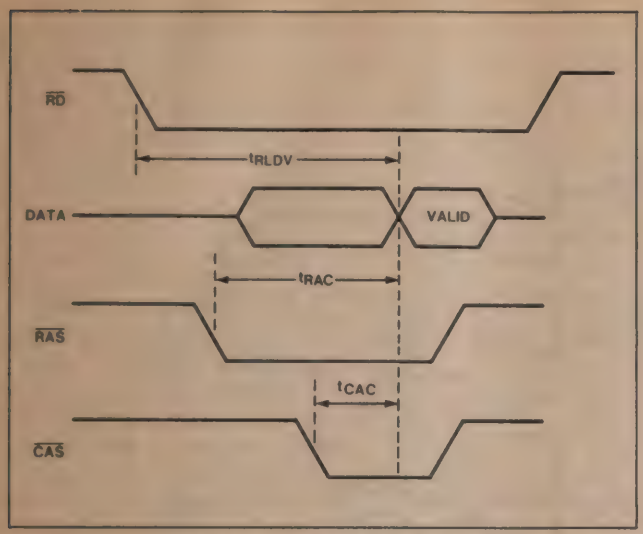


Figure 9. Read Access Time

may increase the number of WAIT states generated by the CPU.

If the $\overline{WE}$ output is externally delayed beyond the $\overline{CAS}$ active transition, then the RAM will use the falling edge of $\overline{WE}$ to strobe the write data into the RAM. This $\overline{WE}$ transition should not occur too late during the CAS active transition, or else the $\overline{WE}$ to $\overline{CAS}$ requirements of the RAM will not be met.

The $\overline{RAS}_{0-3}$, $\overline{CAS}$, $\overline{OUT}_{0-7}$, and $\overline{WE}$ outputs contain on-chip series damping resistors (typically 20Ω) to minimize overshoot.

Some dynamic RAMs require more than 2.4V V_{IH}. Noise immunity may be improved for these RAMs by adding pull-up resistors to the 8203's outputs. Intel RAMs do not require pull-up resistors.

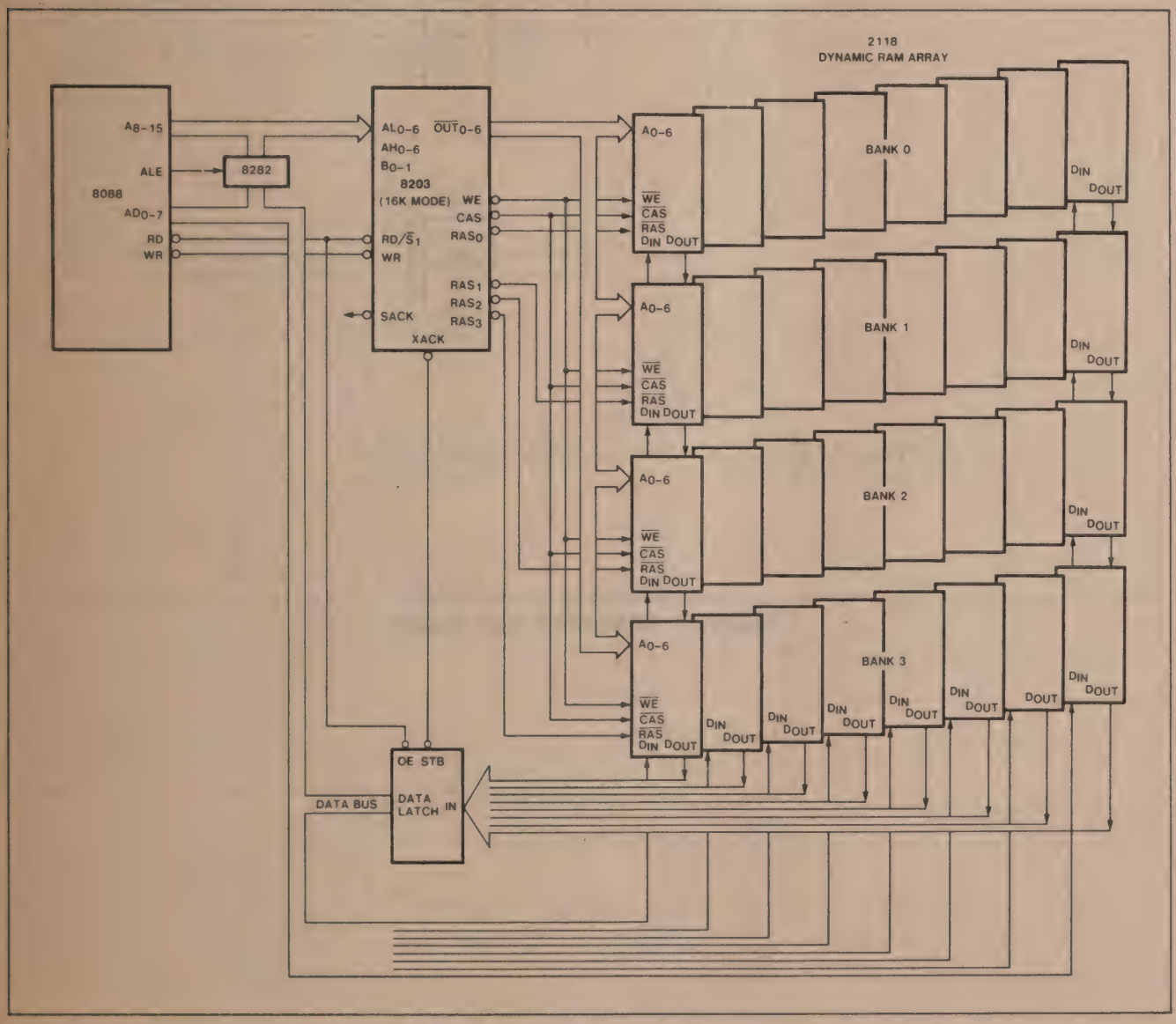


Figure 10. Typical 8088 System

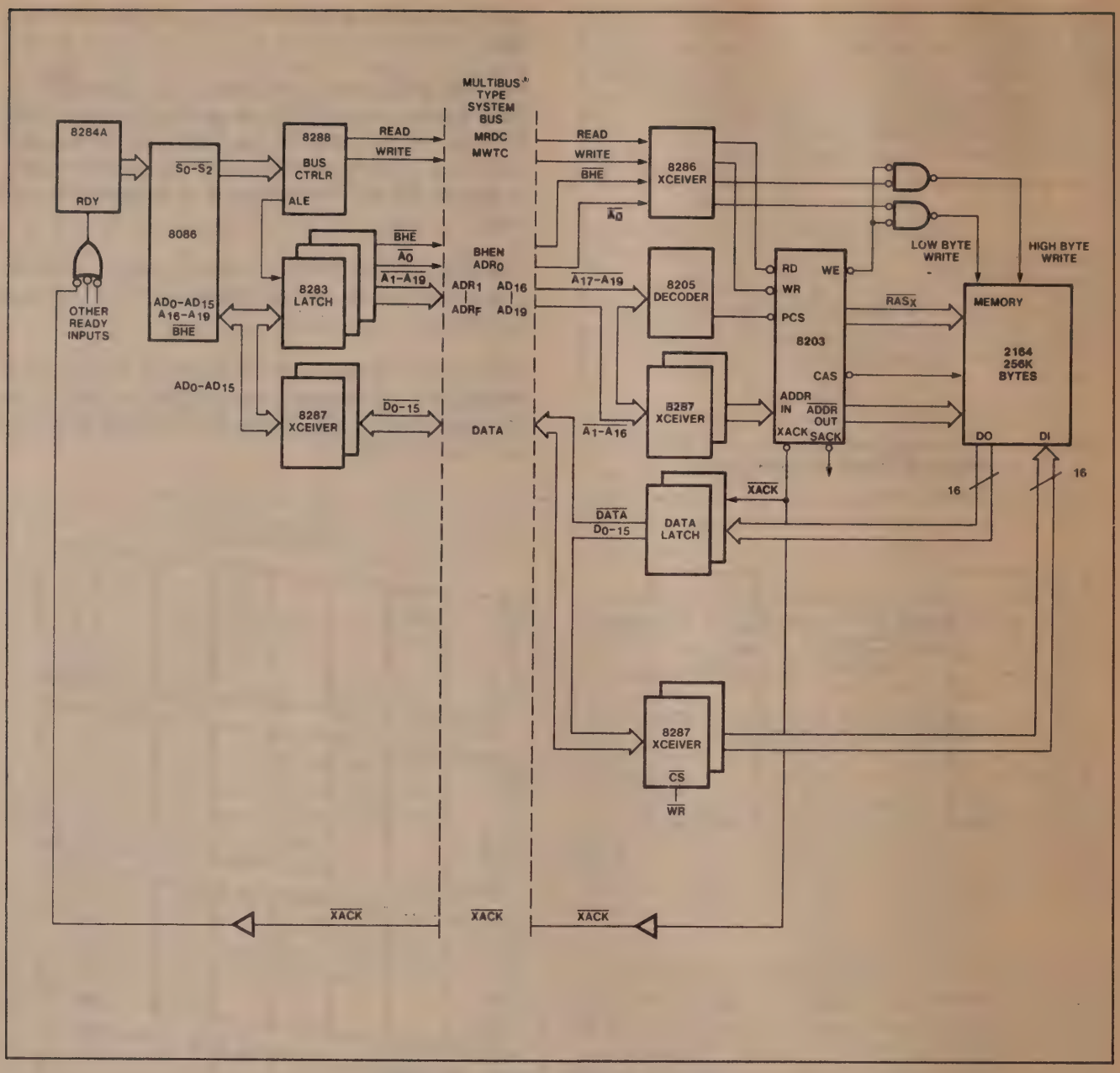


Figure 11. 8086/256K Byte System

A.C. CHARACTERISTICS

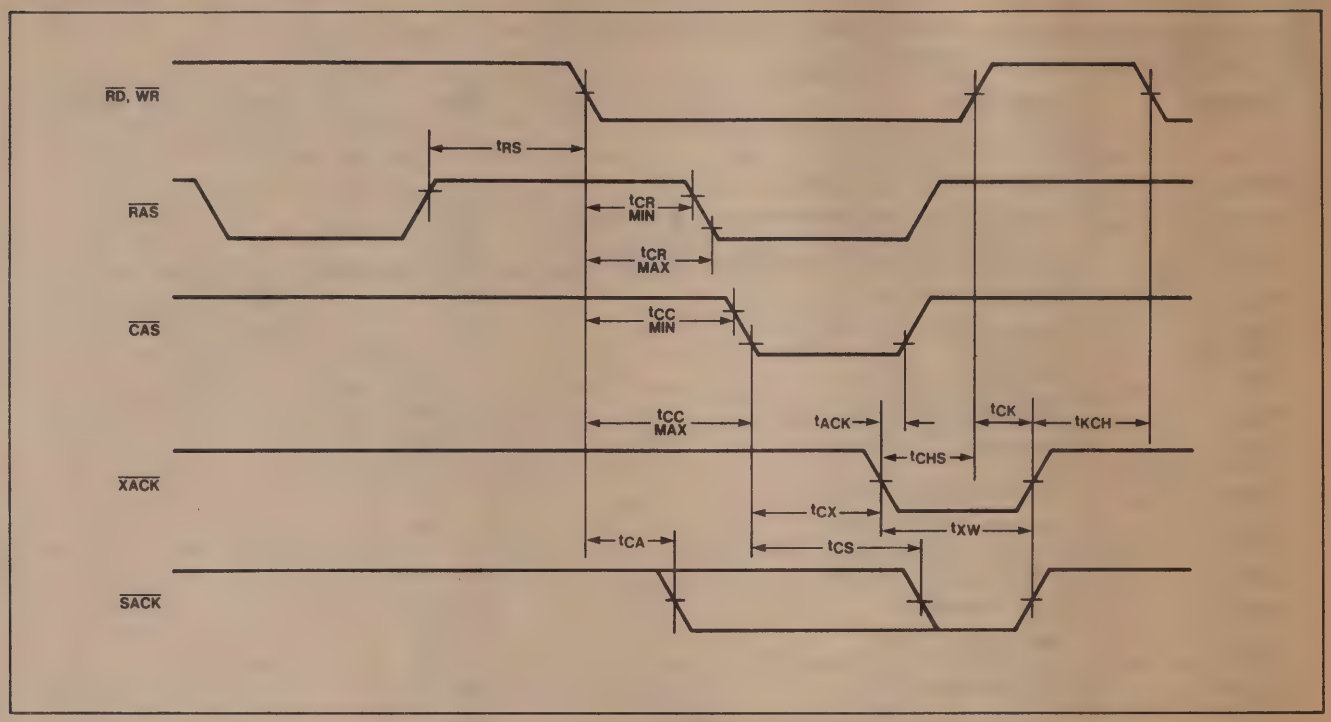
T_J = 0°C to 70°C; V_{CC} = 5V ± 10% (5.0V ± 5% for 8203-3); GND = 0V

Measurements made with respect to $\overline{RAS}_0$-$\overline{RAS}_3$, $\overline{CAS}$, $\overline{WE}$, $\overline{OUT}_0$-$\overline{OUT}_6$ are at 2.4V and 0.8V. All other pins are measured at 1.5V. All times are in nsec.

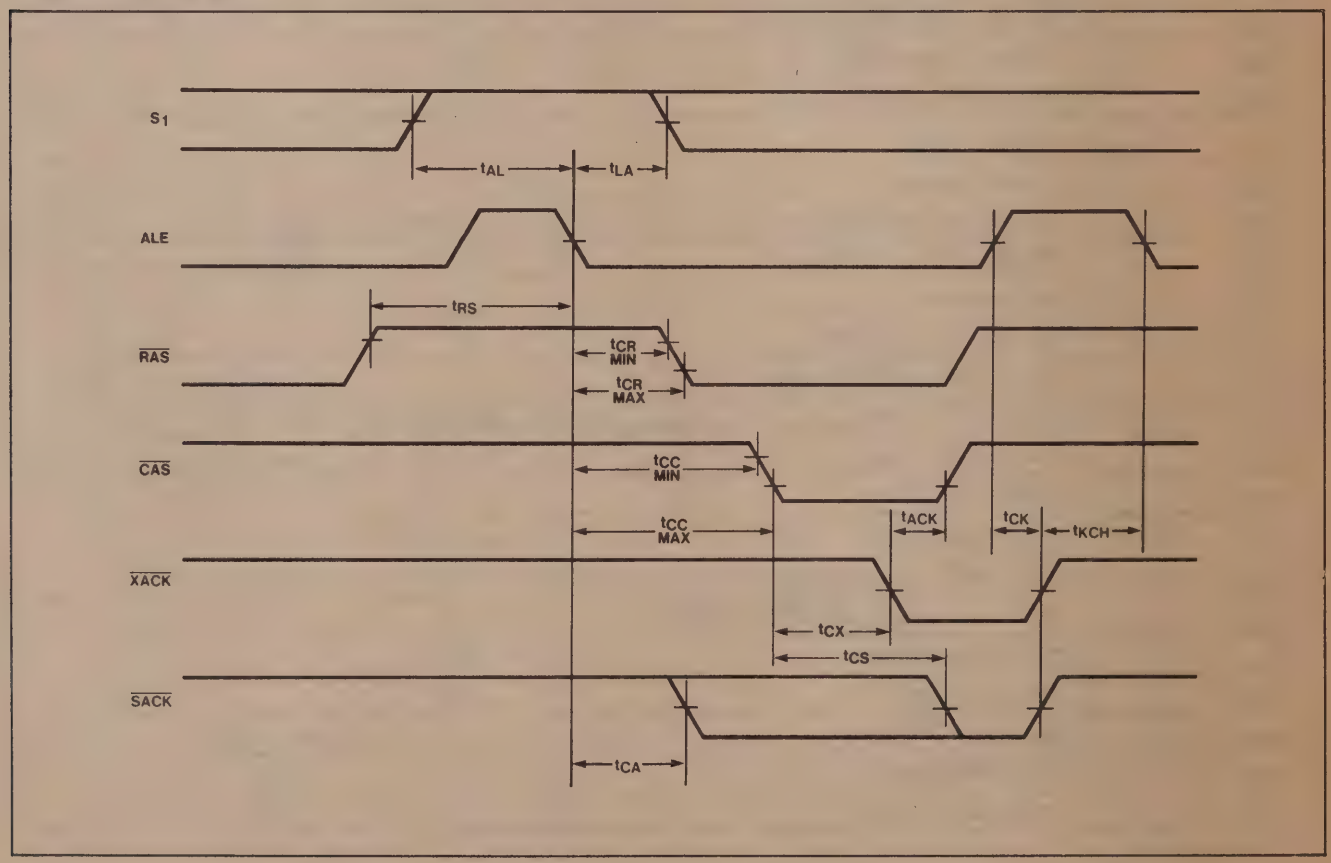
Symbol	Parameter	Min	Max	Notes
t_P	Clock Period	40	54	
t_{PH}	External Clock High Time	20		
t_{PL}	External Clock Low Time—above (>) 20 mHz	17		
t_{PL}	External Clock Low Time—below (≤) 20 mHz	20		
t_{RC}	Memory Cycle Time	10tp − 30	12tp	4, 5
t_{REF}	Refresh Time (128 cycles)	264tp	288tp	
t_{RP}	$\overline{RAS}$ Precharge Time	4tp − 30		
t_{RSH}	$\overline{RAS}$ Hold After $\overline{CAS}$	5tp − 30		3
t_{ASR}	Address Setup to $\overline{RAS}$	tp − 30		3
t_{RAH}	Address Hold From $\overline{RAS}$	tp − 10		3
t_{ASC}	Address Setup to $\overline{CAS}$	tp − 30		3
t_{CAH}	Address Hold from $\overline{CAS}$	5tp − 20		3
t_{CAS}	$\overline{CAS}$ Pulse Width	5tp − 10		
t_{WCS}	$\overline{WE}$ Setup to $\overline{CAS}$	tp − 40		
t_{WCH}	$\overline{WE}$ Hold After $\overline{CAS}$	5tp − 35		8
t_{RS}	$\overline{RD}$, $\overline{WR}$, ALE, REFRQ delay from $\overline{RAS}$	5tp		2, 6
t_{MRP}	$\overline{RD}$, $\overline{WR}$ setup to $\overline{RAS}$	0		5
t_{RMS}	REFRQ setup to $\overline{RD}$, $\overline{WR}$	2tp		6
t_{RMP}	REFRQ setup to $\overline{RAS}$	2tp		5
t_{PCS}	$\overline{PCS}$ Setup to $\overline{RD}$, $\overline{WR}$, ALE	20		
t_{AL}	S1 Setup to ALE	15		
t_{LA}	S1 Hold from ALE	30		
t_{CR}	$\overline{RD}$, $\overline{WR}$, ALE to $\overline{RAS}$ Delay	tp + 30	2tp + 70	2
t_{CC}	$\overline{RD}$, $\overline{WR}$, ALE to $\overline{CAS}$ Delay	3tp + 25	4tp + 85	2
t_{SC}	CMD Setup to Clock	15		1
t_{MRS}	$\overline{RD}$, $\overline{WR}$ setup to REFRQ	5		2
t_{CA}	$\overline{RD}$, $\overline{WR}$, ALE to $\overline{SACK}$ Delay		2tp + 47	2, 9
t_{CX}	$\overline{CAS}$ to $\overline{XACK}$ Delay	5tp − 25	5tp + 20	
t_{CS}	$\overline{CAS}$ to $\overline{SACK}$ Delay	5tp − 25	5tp + 40	2, 10
t_{ACK}	$\overline{XACK}$ to $\overline{CAS}$ Setup	10		
t_{XW}	$\overline{XACK}$ Pulse Width	tp − 25		7
t_{CK}	$\overline{SACK}$, $\overline{XACK}$ turn-off Delay		35	
t_{KCH}	CMD Inactive Hold after $\overline{SACK}$, $\overline{XACK}$	10		
t_{LL}	REFRQ Pulse Width	20		
t_{CHS}	CMD Hold Time	30		11
t_{RFR}	REFRQ to $\overline{RAS}$ Delay		4tp + 100	6
t_{WW}	$\overline{WR}$ to $\overline{WE}$ Delay	0	50	8
t_{AD}	CPU Address Delay	0	40	3

WAVEFORMS

Normal Read or Write Cycle

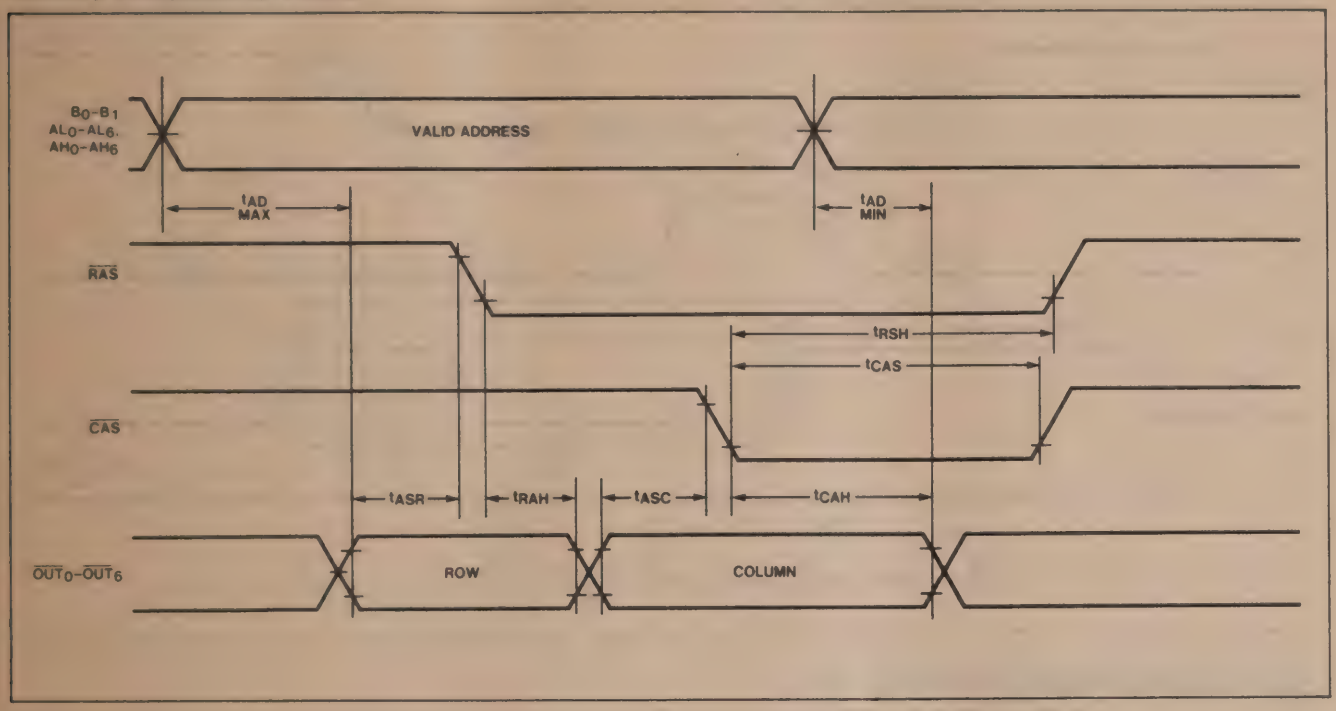


Advanced Read Mode

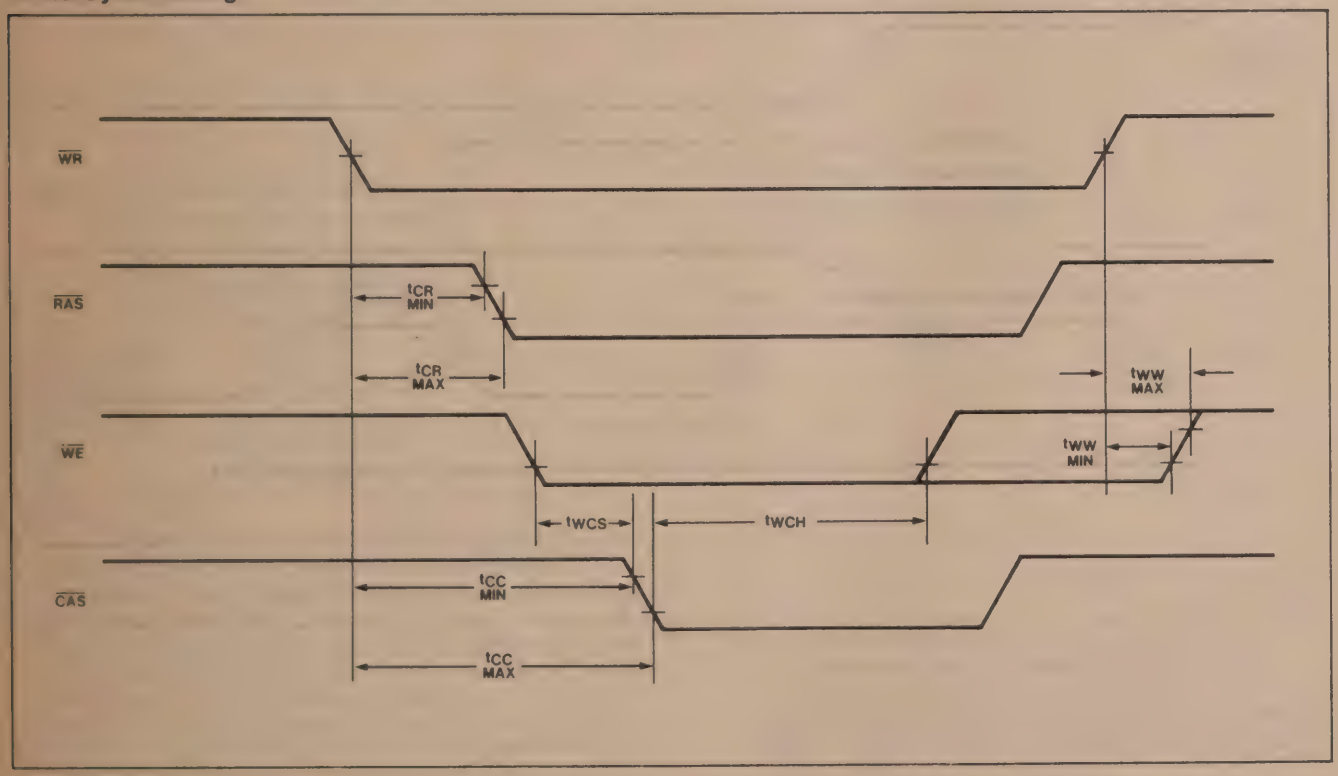


WAVEFORMS (cont'd)

Memory Compatibility Timing

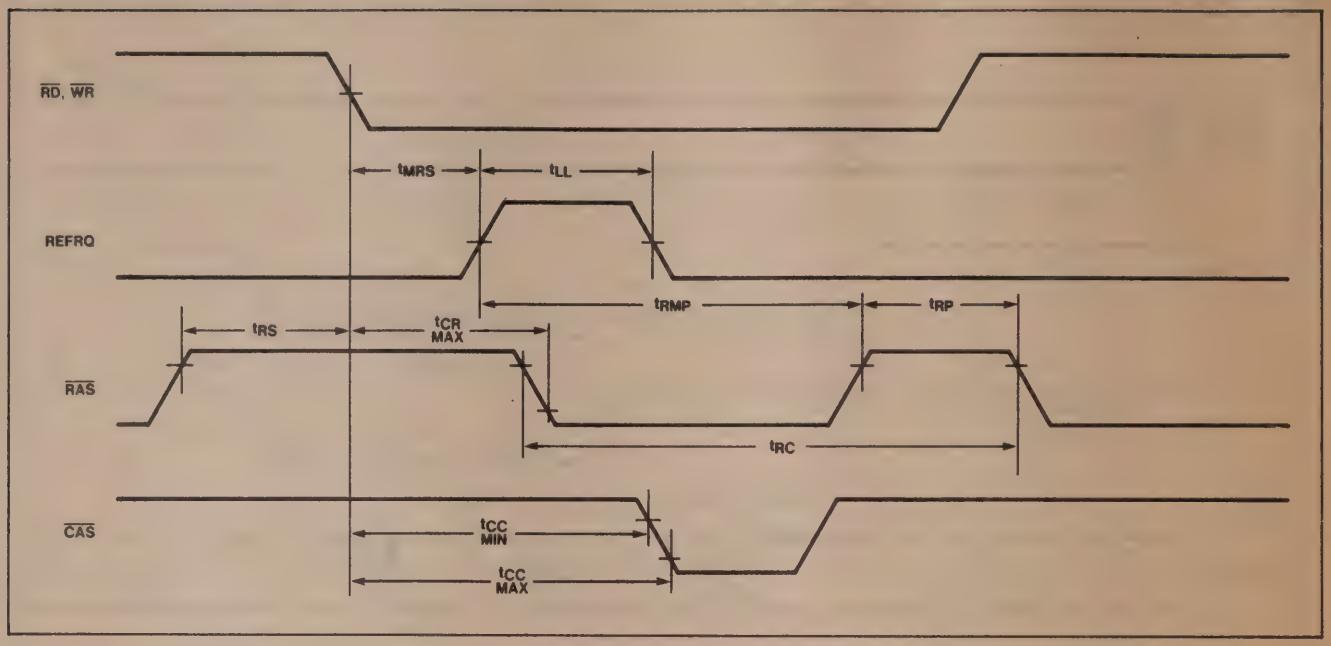


Write Cycle Timing

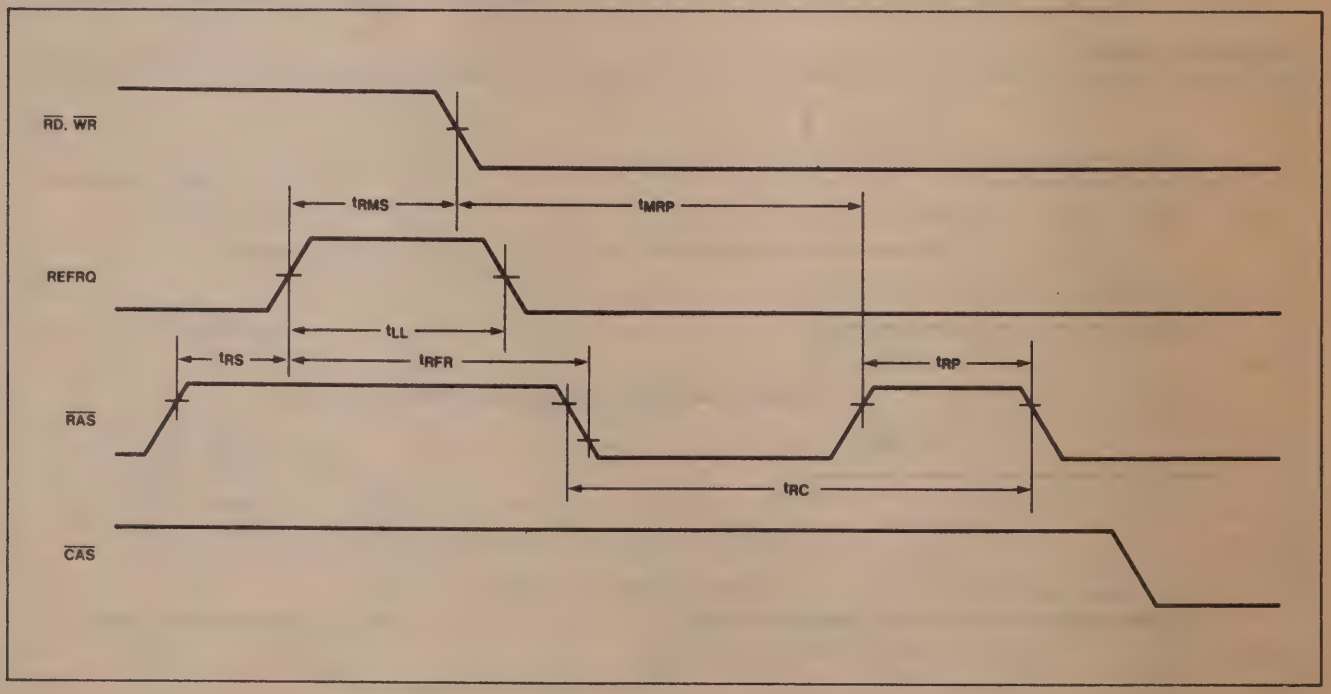


WAVEFORMS (cont'd)

Read or Write Followed By External Refresh



External Refresh Followed By Read or Write



WAVEFORMS (cont'd)

Clock And System Timing

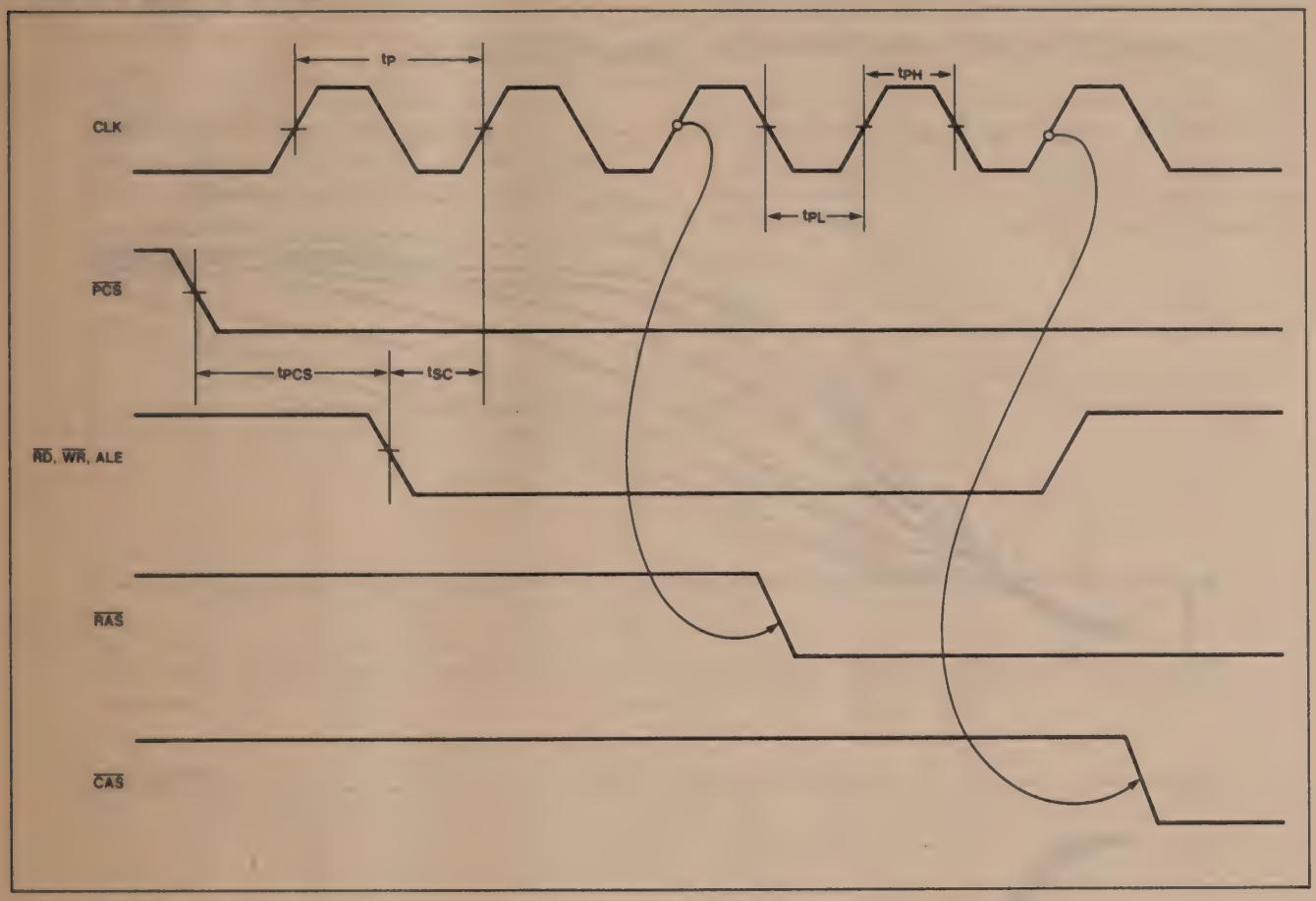


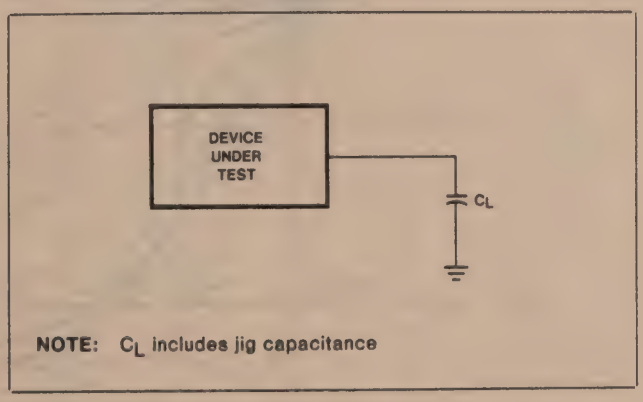
Table 2. 8203 Output Loading. All specifications are for the Test Load unless otherwise noted.

Pin	Test Load
$\overline{SACK}$, $\overline{XACK}$	C_L = 30 pF
$\overline{OUT}_0$-$\overline{OUT}_6$	C_L = 160 pF
$\overline{RAS}_0$-$\overline{RAS}_3$	C_L = 60 pF
$\overline{WE}$	C_L = 224 pF
$\overline{CAS}$	C_L = 320 pF

NOTES:

1. t_{SC} is a reference point only. ALE, $\overline{RD}$, $\overline{WR}$, and REFRQ inputs do not have to be externally synchronized to 8203 clock.
2. If t_{RS} min and t_{MRS} min are met then t_{CA}, t_{CR}, and t_{CC} are valid, otherwise t_{CS} is valid.
3. t_{ASR}, t_{RAH}, t_{ASC}, t_{CAH}, and t_{RSH} depend upon B0-B1 and CPU address remaining stable throughout the memory cycle. The address inputs are not latched by the 8203.
4. For back-to-back refresh cycles, t_{RC} max = 13tp
5. t_{RC} max is valid only if t_{RMP} min is met (READ, WRITE followed by REFRESH) or t_{MRP} min is met (REFRESH followed by READ, WRITE).
6. t_{RFR} is valid only if t_{RS} min and t_{RMS} min are met.
7. t_{XW} min applies when $\overline{RD}$, $\overline{WR}$ has already gone high. Otherwise $\overline{XACK}$ follows $\overline{RD}$, $\overline{WR}$.
8. $\overline{WE}$ goes high according to t_{WCH} or t_{WW}, whichever occurs first.

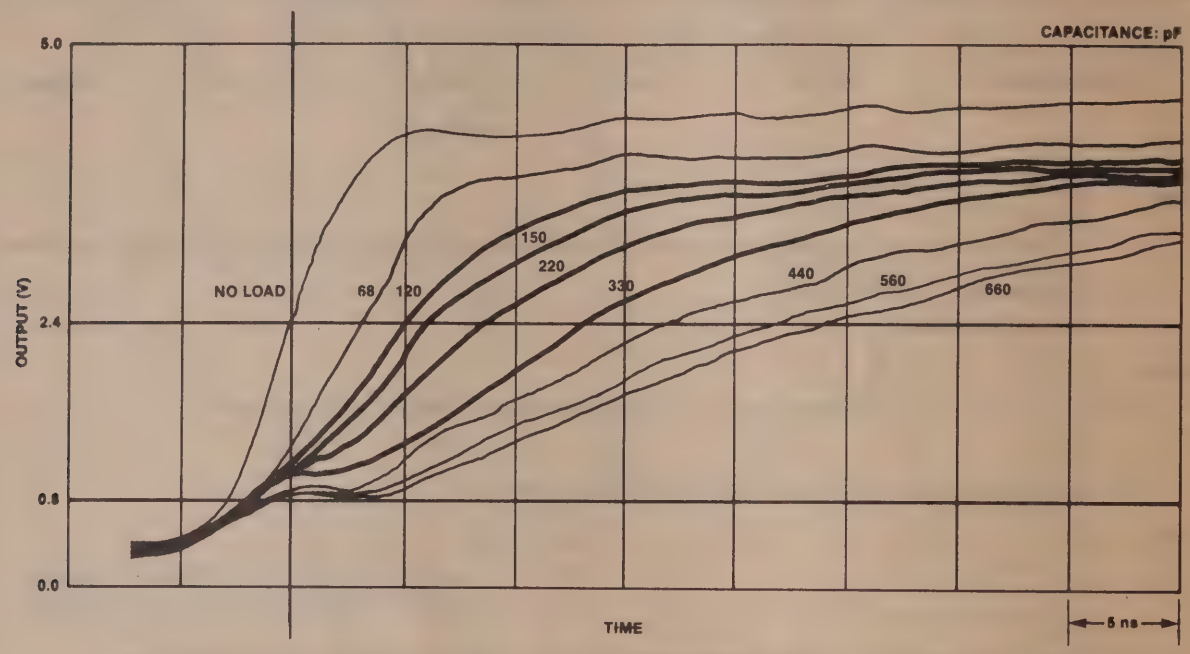
A.C. TESTING LOAD CIRCUIT

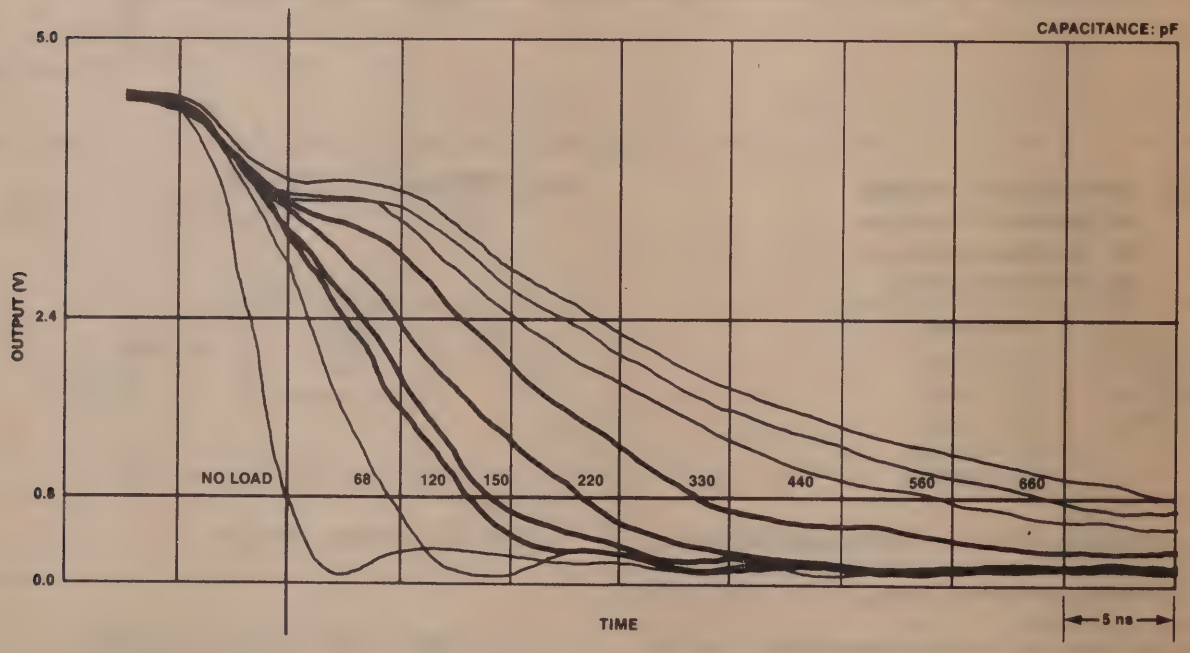


9. t_{CA} applies only when in normal $\overline{SACK}$ mode.de.
10. t_{CS} applies only when in delayed $\overline{SACK}$ mode.
11. t_{CHS} must be be met only to ensure a $\overline{SACK}$ active pulse when in delayed $\overline{SACK}$ mode. $\overline{XACK}$ will always be activated for at least t_{XW} (t_P−25 nS). Violating t_{CHS} min does not otherwise affect device operation.

The typical rising and falling characteristic curves for the $\overline{OUT}$, $\overline{RAS}$, $\overline{CAS}$ and $\overline{WE}$ output buffers can be used to determine the effects of capacitive loading on the A.C. Timing Parameters. Using this design tool in conjunction with the timing waveforms, the designer can determine typical timing shifts based on system capacitive load.

A.C. CHARACTERISTICS FOR DIFFERENT CAPACITIVE LOADS





NOTE:
Use the Test Load as the base capacitance for estimating timing shifts for system critical timing parameters.

MEASUREMENT CONDITIONS:

$T_A = 25°C$
$V_{CC} = +5V$
$t_p = 50$ ns

Pins not measured are loaded with the Test Load capacitance

Example: Find the effect on t_{CR} and t_{CC} using 32 2164 Dynamic RAMs configured in 2 banks.

1. Determine the typical RAS and CAS capacitance:
   From the data sheet RAS = 5 pF and CAS = 5 pF.
   ∴ RAS load = 80 pF + board capacitance.
   CAS load = 160 pF + board capacitance.
   Assume 2 pF/in (trace length) for board capacitance and for this example 4 inches for RAS and 8 inches for CAS.

2. From the waveform diagrams, we determine that the falling edge timing is needed for t_{CR} and t_{CC}. Next find the curve that *best* approximates the test load; i.e., 68 pF for RAS and 330 pF for CAS.

3. If we use 88 pF for RAS loading, then t_{CR} (min.) spec should be increased by about 1 ns, and t_{CR} (max.) spec should be increased by *about* 2 ns. Similarly if we use 176 pF for CAS, then t_{CC} (min.) should decrease by 3 ns and t_{CC} (max.) should decrease by about 7 ns.

8206
ERROR DETECTION AND CORRECTION UNIT

- Detects All Single Bit, Double Bit and Most Multiple Bit Errors
- Corrects All Single Bit Errors
-

3 Selections	8206-1	8206	8206-2
Detection	35ns	42ns	57ns
Correction	55ns	67ns	74ns

- 8206-2 Timing Supports Single 8206 8MHz iAPX 186, 188, 86, 88 and 8207-8 Systems
- Syndrome Outputs for Error Logging
- Separate Input and Output Busses—No Timing Strobes Required
- Expandable to Handle 80 Bit Memories
- Supports Read With and Without Correction, Writes, Partial (Byte) Writes, and Read-Modify-Writes
- HMOS III Technology for Low Power
- 68 Pin Leadless JEDEC Package
- 68 Pin Grid Array Package

The HMOS 8206 Error Detection and Correction Unit is a high-speed device that provides error detection and correction for memory systems (static and dynamic) requiring high reliability and performance. Each 8206 handles 8 or 16 data bits and up to 8 check bits. 8206's can be cascaded to provide correction and detection for up to 80 bits of data. Other 8206 features include the ability to handle byte writes, memory initialization, and error logging.

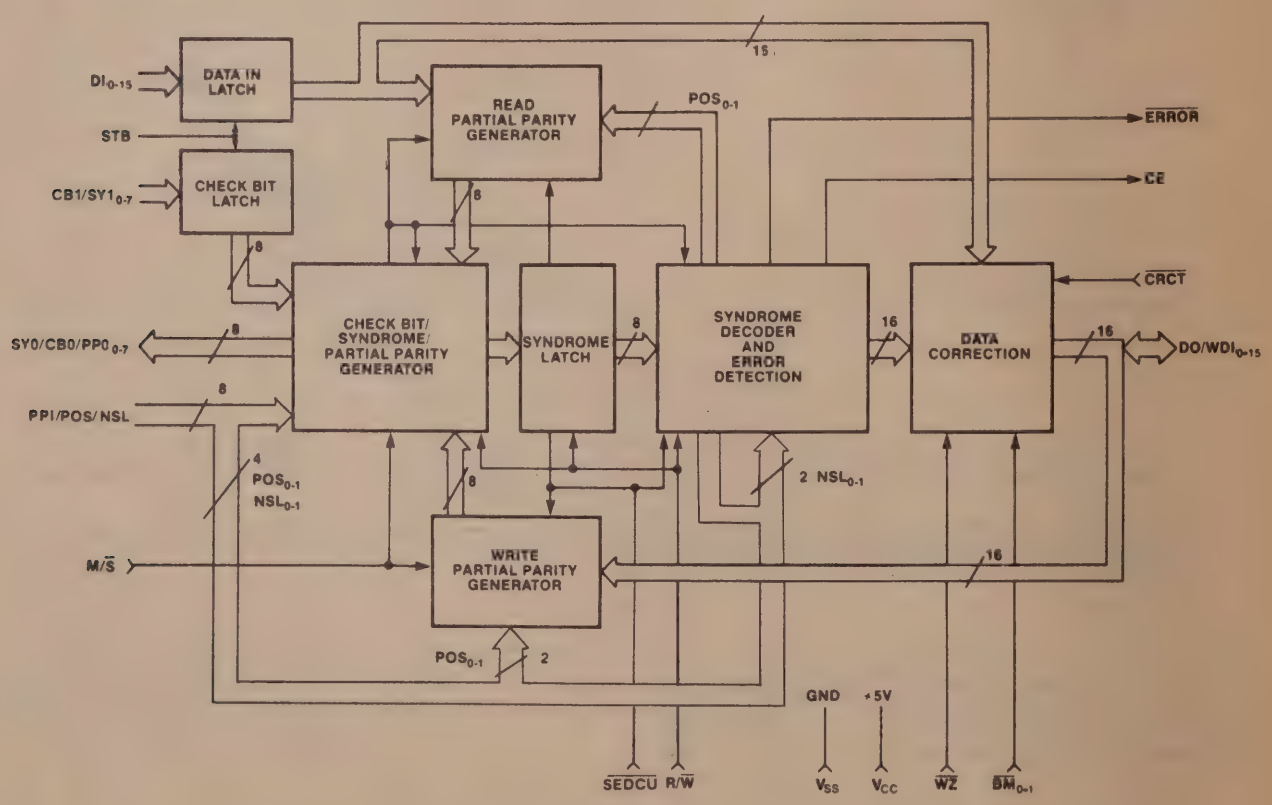


Figure 1. 8206 Block Diagram

Intel Corporation Assumes No Responsibility for the Use of Any Circuitry Other Than Circuitry Embodied in an Intel Product. No Other Circuit Patent Licenses are Implied.
© INTEL CORPORATION, 1982.

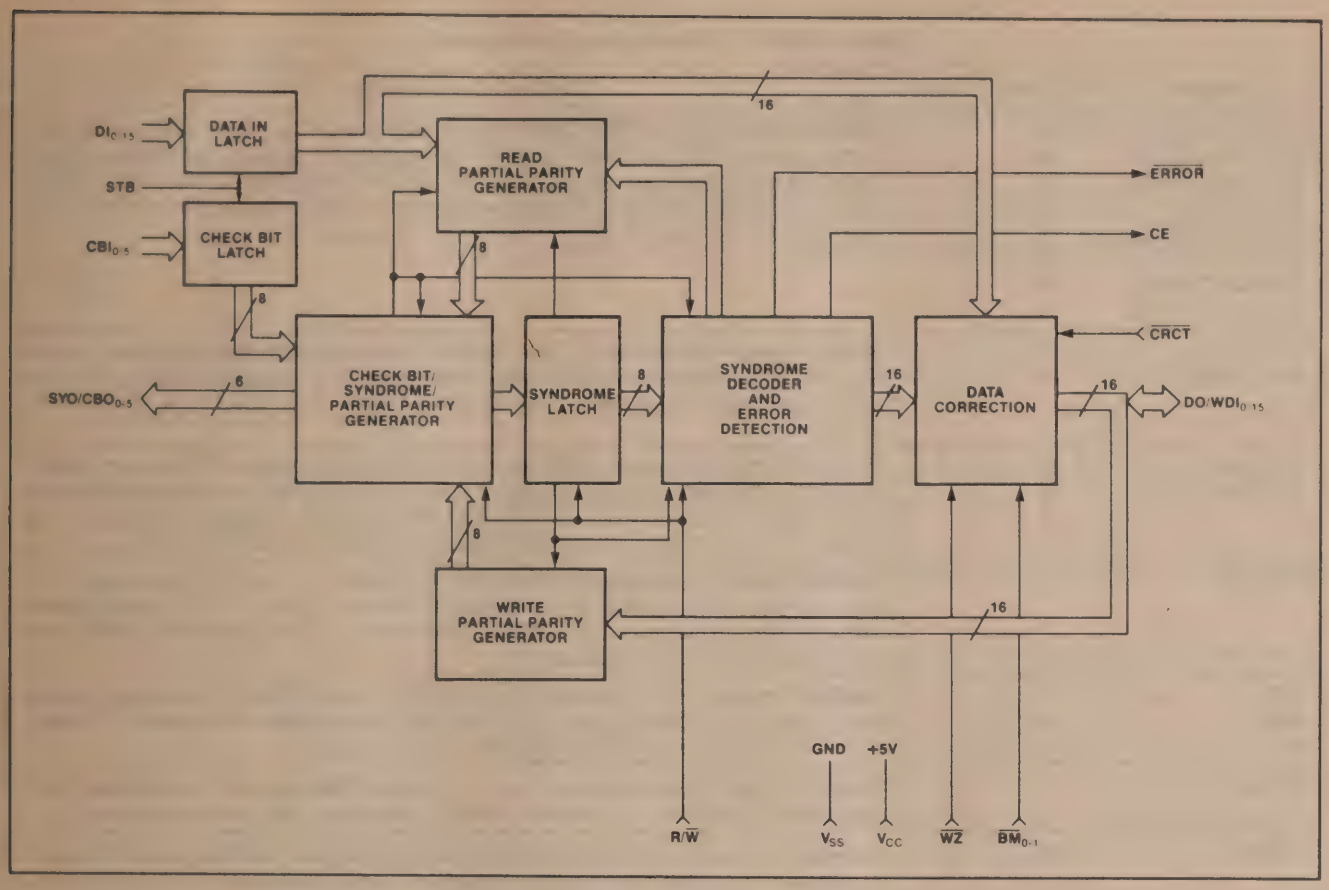


Figure 2. 8206-2 Block Diagram

Table 1. 8206 Pin Description

Symbol	Pin No.	Type	Name and Function
DI_{0-15}	1, 68-61, 59-53	I	**Data In:** These inputs accept a 16 bit data word from RAM for error detection and/or correction.
CBI/SYI_0	5	I	**Check Bits In/Syndrome In:** In a single 8206 system, or in the master in a multi-8206 system, these inputs accept the check bits (5 to 8) from the RAM. In a single 8206 16 bit system, CBI_{0-5} are used. In slave 8206's these inputs accept the syndrome from the master.
CBI/SYI_1	6	I	
CBI/SYI_2	7	I	
CBI/SYI_3	8	I	
CBI/SYI_4	9	I	
CBI/SYI_5	10	I	
CBI/SYI_6	11	I	
CBI/SYI_7	12	I	
DO/WDI_0	51	I/O	**Data Out/Write Data In:** In a read cycle, data accepted by DI_{0-15} appears at these outputs corrected if $\overline{CRCT}$ is low, or uncorrected if $\overline{CRCT}$ is high. The $\overline{BM}$ inputs must be high to enable the output buffers during the read cycle. In a write cycle, data to be written into the RAM is accepted by these inputs for computing the write check bits. In a partial-write cycle, the byte not to be modified appears at either DO_{0-7} if $B\overline{M}_0$ is high, or DO_{8-15} if $B\overline{M}_1$ is high, for writing to the RAM. When $\overline{WZ}$ is active, it causes the 8206 to output all zeros at DO_{0-15}, with the proper write check bits on CBO.
DO/WDI_1	50	I/O	
DO/WDI_2	49	I/O	
DO/WDI_3	48	I/O	
DO/WDI_4	47	I/O	
DO/WDI_5	46	I/O	
DO/WDI_6	45	I/O	
DO/WDI_7	44	I/O	
DO/WDI_8	42	I/O	
DO/WDI_9	41	I/O	
DO/WDI_{10}	40	I/O	
DO/WDI_{11}	39	I/O	
DO/WDI_{12}	38	I/O	
DO/WDI_{13}	37	I/O	
DO/WDI_{14}	36	I/O	
DO/WDI_{15}	35	I/O	

Table 1. 8206 Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
SYO/CBO/PPO_0 SYO/CBO/PPO_1 SYO/CBO/PPO_2 SYO/CBO/PPO_3 SYO/CBO/PPO_4 SYO/CBO/PPO_5 SYO/CBO/PPO_6 SYO/CBO/PPO_7	23 24 25 27 28 29 30 31	O O O O O O O O	**Syndrome Out/Check Bits Out/Partial Parity Out:** In a single 8206 system, or in the master in a multi-8206 system, the syndrome appears at these outputs during a read. During a write, the write check bits appear. In slave 8206's the partial parity bits used by the master appear at these outputs. The syndrome is latched (during read-modify-writes) by R/$\overline{W}$ going low.
PPI_0/POS_0 PPI_1/POS_1	13 14	I I	**Partial Parity In/Position:** In the master in a multi-8206 system, these inputs accept partial parity bits 0 and 1 from the slaves. In a slave 8206 these inputs inform it of its position within the system (1 to 4). Not used in a single 8206 system.
PPI_2/NSL_0 PPI_3/NSL_1	15 16	I I	**Partial Parity In/Number of Slaves:** In the master in a multi-8206 system, these inputs accept partial parity bits 2 and 3 from the slaves. In a multi-8206 system these inputs are used in slave number 1 to tell it the total number of slaves in the system (1 to 4). Not used in other slaves or in a single 8206 system.
PPI_4/CE	17	I/O	**Partial Parity In/Correctable Error:** In the master in a multi-8206 system this pin accepts partial parity bit 4. In slave number 1 only, or in a single 8206 system, this pin outputs the correctable error flag. CE is latched by R/$\overline{W}$ going low. Not used in other slaves.
PPI_5 PPI_6 PPI_7	18 19 20	I I I	**Partial Parity In:** In the master in a multi-8206 system these pins accept partial parity bits 5 to 7. The number of partial parity bits equals the number of check bits. Not used in single 8206 systems or in slaves.
$\overline{ERROR}$	22	O	**Error:** This pin outputs the error flag in a single 8206 system or in the master of a multi-8206 system. It is latched by R/$\overline{W}$ going low. Not used in slaves.
$\overline{CRCT}$	52	I	**Correct:** When low this pin causes data correction during a read or read-modify-write cycle. When high, it causes error correction to be disabled, although error checking is still enabled.
STB	2	I	**Strobe:** STB is an input control used to strobe data at the DI inputs and check-bits at the CBI/SYI inputs. The signal is active high to admit the inputs. The signals are latched by the high-to-low transition of STB.
$\overline{BM_0}$ $\overline{BM_1}$	33 32	I I	**Byte Marks:** When high, the Data Out pins are enabled for a read cycle. When low, the Data Out buffers are tristated for a write cycle. $\overline{BM_0}$ controls DO_{0-7}, while $\overline{BM_1}$ controls DO_{8-15}. In partial (byte) writes, the byte mark input is low for the new byte to be written.
R/$\overline{W}$	21	I	**Read/Write:** When high this pin causes the 8206 to perform detection and correction (if $\overline{CRCT}$ is low). When low, it causes the 8206 to generate check bits. On the high-to-low transition the syndrome is latched internally for read-modify-write cycles.
$\overline{WZ}$	34	I	**Write Zero:** When low this input overrides the $\overline{BM_{0-1}}$ and R/$\overline{W}$ inputs to cause the 8206 to output all zeros at DO_{0-15} with the corresponding check bits at CBO_{0-7}. Used for memory initialization.
M/$\overline{S}$	4	I	**Master/Slave:** Input tells the 8206 whether it is a master (high) or a slave (low).
$\overline{SEDCU}$	3	I	**Single EDC Unit:** Input tells the master whether it is operating as a single 8206 (low) or as the master in a multi-8206 system (high). Not used in slaves.
V_{CC}	60	I	**Power Supply:** +5V
V_{SS}	26	I	**Logic Ground**
V_{SS}	43	I	**Output Driver Ground**

Table 2. 8206-2 Pin Description Differences over the 8206.

Symbol	Pin	Type	Name and Function
CBI_{0-5}	5-10	I	**Check Bits In:** In an 8206-2 system, these inputs accept the check bits (5 to 6) from the RAM.
SYO/CBO_0 SYO/CBO_1 SYO/CBO_2 SYO/CBO_3 SYO/CBO_4 SYO/CBO_5	23 24 25 27 28 29	O O O O O O	**Syndrome Out/Check Bits Out:** In an 8206-2 system, the syndrome appears at these outputs during a read. During a write, the write check bits appear. The syndrome is latched (during read-modify-writes) by $R/\overline{W}$ going low.
CE	17	O	**Correctable Error:** In an 8206-2 system, this pin outputs the correctable error flag. CE is latched by $R/\overline{W}$ going low.
$\overline{WZ}$	34	I	**Write Zero:** When low this input overrides the BM_{0-1} and R/W inputs to cause the 8206-2 to output all zeros at DO_{0-15} with the corresponding check bits at CBO_{0-5}. Used for memory initialization.
Strap High	4	I	Must be tied High.
Strap Low	3	I	Must be tied Low.
N.C.	11-16 18-20	I	**Note:** These pins have internal pull-up resistors but if possible should be tied high or low.
N.C.	30, 31	O	**Note:** These are no connect pins and should be left open.

FUNCTIONAL DESCRIPTION

The 8206 Error Detection and Correction Unit provides greater memory system reliability through its ability to detect and correct memory errors. It is a single chip device that can detect and correct all single bit errors and detect all double bit and some higher multiple bit errors. Some other odd multiple bit errors (e.g., 5 bits in error) are interpreted as single bit errors, and the CE flag is raised. While some even multiple bit errors (e.g., 4 bits in error) are interpreted as no error, most are detected as double bit errors. This error handling is a function of the number of check bits used by the 8206 (see Figure 2) and the specific Hamming code used. Errors in check bits are not distinguished from errors in a word.

For more information on error correction codes, see Intel Application Notes AP-46 and AP-73.

DATA WORD BITS	CHECK BITS
8	5
16	6
24	6
32	7
40	7
48	8
56	8
64	8
72	8
80	8

Figure 3. Number of Check Bits Used by 8206

A single 8206 or 8206-2 handles 8 or 16 bits of data, and up to 5 8206's can be cascaded in order to handle data paths of 80 bits. For a single 8206 8 bit system, the DI_{8-15}, DO/WDI_{8-15} and BM_1 inputs are grounded. See the Multi-Chip systems section for information on 24-80 bit systems.

The 8206 has a "flow through" architecture. It supports two kinds of error correction architecture: 1) Flow-through, or correct-always; and 2) Parallel, or check-only. There are two separate 16-pin busses, one to accept data from the RAM (DI) and the other to deliver corrected data to the system bus (DO/WDI). The logic is entirely combinatorial during a read cycle. This is in contrast to an architecture with only one bus, with bidirectional bus drivers that must first read the data and then be turned around to output the corrected data. The latter architecture typically requires additional hardware (latches and/or transceivers) and may be slower in a system due to timing skews of control signals.

READ CYCLE

With the R/$\overline{W}$ pin high, data is received from the RAM outputs into the DI pins where it is optionally latched by the STB signal. Check bits are generated from the data bits and compared to the check bits read from the RAM into the CBI pins. If an error is detected the $\overline{ERROR}$ flag is activated and the correctable error flag (CE) is used to inform the system whether the error was correctable or not. With the $\overline{BM}$ inputs high, the word appears corrected at the DO pins if the error was correctable, or unmodified if the error was uncorrectable.

If more than one 8206 is being used, then the check bits are read by the master. The slaves generate a partial parity output (PPO) and pass it to the master. The master 8206 then generates and returns the syndrome to the slaves (SYO) for correction of the data.

The 8206 may alternatively be used in a "check-only" mode with the $\overline{CRCT}$ pin left high. With the correction facility turned off, the propagation delay from memory outputs to 8206 outputs is significantly shortened. In this mode the 8206 issues an $\overline{ERROR}$ flag to the CPU, which can then perform one of several options: lengthen the current cycle for correction, restart the instruction, perform a diagnostic routine, etc.

A syndrome word, five to eight bits in length and containing all necessary information about the existence and location of an error, is made available to the system at the SYO_{0-7} pins. Error logging may be accomplished by latching the syndrome and the memory address of the word in error.

WRITE CYCLE

For a full write, in which an entire word is written to memory, the data is written directly to the RAM, bypassing the 8206. The same data enters the 8206 through the WDI pins where check bits are generated. The Byte Mark inputs must be low to tristate the DO drivers. The check bits, 5 to 8 in number, are then written to the RAM through the CBO pins for storage along with the data word. In a multi-chip system, the master writes the check bits using partial parity information from the slaves.

In a partial write, part of the data word is overwritten, and part is retained in memory. This is accomplished by performing a read-modify-write cycle. The complete old word is read into the 8206 and corrected, with the syndrome internally latched by R/$\overline{W}$ going low. Only that part of the word not to be modified is output onto the DO pins, as controlled by the Byte Mark inputs. That portion of the word to be overwritten is supplied by the system bus. The 8206 then calculates check bits for the new word, using the byte from the previous read and the new byte from the system bus, and writes them to the memory.

READ-MODIFY-WRITE CYCLES

Upon detection of an error the 8206 may be used to correct the bit in error in memory. This reduces the probability of getting multiple-bit errors in subsequent read cycles. This correction is handled by executing read-modify-write cycles.

The read-modify-write cycle is controlled by the R/$\overline{W}$ input. After (during) the read cycle, the system dynamic RAM controller or CPU examines the 8206 $\overline{ERROR}$ and CE outputs to determine if a correctable error occurred. If it did, the dynamic RAM controller or CPU forces R/$\overline{W}$ low, telling the 8206 to latch the generated syndrome and drive the corrected check bits onto the CBO outputs. The corrected data is available on the DO pins. The DRAM controller then writes the corrected data and corresponding check bits into memory.

The 8206 may be used to perform read-modify-writes in one or two RAM cycles. If it is done in two cycles, the 8206 latches are used to hold the data and check bits from the read cycle to be used in the following write cycle. The Intel 8207 Advanced Dynamic RAM controller allows read-modify-write cycles in one memory cycle. See the System Environment section.

INITIALIZATION

A memory system operating with ECC requires some form of initialization at system power-up in order to set valid data and check bit information in memory. The 8206 supports memory initialization by the write zero function. By activating the $\overline{WZ}$ pin, the 8206 will write a data pattern of zeros and the associated check bits in the current write cycle. By thus writing to all memory at power-up, a controller can set memory to valid data and check bits. Massive memory failure, as signified by both data and check bits all ones or zeros, will be detected as an uncorrectable error.

MULTI-CHIP SYSTEMS

A single 8206 handles 8 or 16 bits of data and 5 or 6 check bits, respectively. Up to 5 8206's can be cascaded for 80 bit memories with 8 check bits.

When cascaded, one 8206 operates as a master, and all others as slaves. As an example, during a read cycle in a 32 bit system with one master and one slave, the slave calculates parity on its portion of the word—"partial parity"—and presents it to the master through the PPO pins. The master combines the partial parity from the slave with the parity it calculated from its own portion of the word to generate the syndrome. The syndrome is then returned by the master to the slave for error correction. In systems with more than one slave the above description continues to apply, except that the partial parity outputs of the slaves must be XOR'd externally. Figure 4 shows the necessary external logic for multi-chip systems. Write and read-modify-write cycles are carried out analogously. See the System Operation section for multi-chip wiring diagrams.

There are several pins used to define whether the 8206 will operate as a master or a slave. Tables 3 and 4 illustrate how these pins are tied.

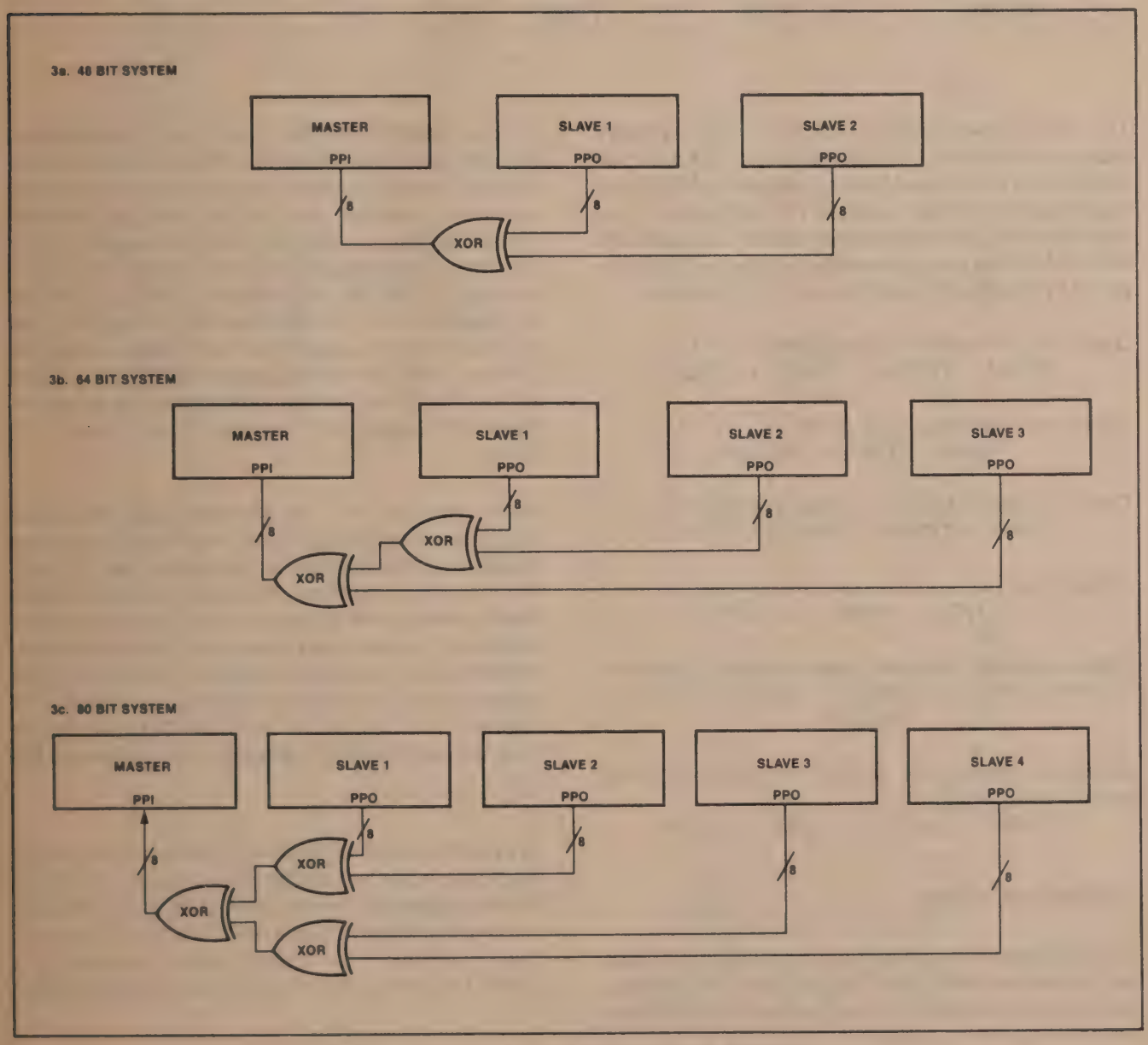


Figure 4. External Logic For Mult-Chip Systems

205220-007

Table 3. Master/Slave Pin Assignments

Pin No.	Pin Name	Master	Slave 1	Slave 2	Slave 3	Slave 4
4	$M/\overline{S}$	+5V	Gnd	Gnd	Gnd	Gnd
3	$\overline{SEDCU}$	+5V	+5V	+5V	+5V	+5V
13	PPI_0/POS_0	PPI	Gnd	+5V	Gnd	+5V
14	PPI_1/POS_1	PPI	Gnd	Gnd	+5V	+5V
15	PPI_2/NSL_0	PPI	*	+5V	+5V	+5V
16	PPI_3/NSL_1	PPI	*	+5V	+5V	+5V

*See Table 3.

NOTE:
Pins 13, 14, 15, 16 have internal pull-up resistors and may be left as N.C. where specified as connecting to +5V.

Table 4. NSL Pin Assignments for Slave 1

	Number of Slaves			
Pin	1	2	3	4
PPI_2/NSL_0	Gnd	+5V	Gnd	+5V
PPI_3/NSL_1	Gnd	Gnd	+5V	+5V

The timing specifications for multi-chip systems must be calculated to take account of the external XOR gating in 3, 4, and 5-chip systems. Let tXOR be the delay for a single external TTL XOR gate. Then the following equations show how to calculate the relevant timing parameters for 2-chip (n=0), 3-chip (n=1), 4-chip (n=2), and 5-chip (n=2) systems:

Data-in to corrected data-out (read cycle) =
TDVSV + TPVSV + TSVQV + ntXOR

Data-in to error flag (read cycle) =
TDVSV + TPVEV + ntXOR

Data-in to correctable error flag (read cycle) =
TDVSV + TPVSV + TSVCV + ntXOR

Write data to check-bits valid (full write cycle) =
TQVQV + TPVSV + ntXOR

Data-in to check-bits valid (read-mod-write cycle) =
TDVSV + TPVSV + TSVQV + TQVQV + TPVSV + 2ntXOR

Data-in to check-bits valid (non-correcting read-modify-write cycle) =
TDVQU + TQVQV + TPVSV + ntXOR

HAMMING CODE

The 8206 uses a modified Hamming code which was optimized for multi-chip EDCU systems. The code is such that partial parity is computed by all 8206's in parallel. No 8206 requires more time for propagation through logic levels than any other one, and hence no one device becomes a bottleneck in the parity operation. However, one or two levels of external TTL XOR gates are required in systems with three to five chips. The code appears in Table 5. The check bits are derived from the table by XORing or XNORing together the bits indicated by 'X's in each row corresponding to a check bit. For example, check bit 0 in the MASTER for data word 1000110101101011 will be "0." It should be noted that the 8206 will detect the gross-error condition of all lows or all highs.

Error correction is accomplished by identifying the bad bit and inverting it. Table 5 can also be used as an error syndrome table by replacing the 'X's with '1's. Each column then represents a different syndrome word, and by locating the column corresponding to a particular syndrome the bit to be corrected may be identified. If the syndrome cannot be located then the error cannot be corrected. For example, if the syndrome word is 00110111, the bit to be corrected is bit 5 in the slave one data word (bit 21).

The syndrome decoding is also summarized in Tables 6 and 7 which can be used for error logging. By finding the appropriate syndrome word (starting with bit zero, the least significant bit), the result is either: 1) no error; 2) an identified (correctable) single bit error; 3) a double bit error; or 4) a multi-bit uncorrectable error.

Table 5. Modified Hamming Code Check Bit Generation

Check bits are generated by XOR'ing (except for the CB0 and CB1 data bits, which are XNOR'ed in the Master) the data bits in the rows corresponding to the check bits. Note there are 6 check bits in a 16-bit system, 7 in a 32-bit system, and 8 in 48-or-more-bit systems.

BYTE NUMBER		0								1								OPERATION
BIT NUMBER		0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	
CHECK BITS	CB0 =	x	x	-	x	-	x	x	-	x	-	-	x	-	x	-	-	XNOR
	CB1 =	x	-	x	-	-	x	-	x	-	x	-	x	x	-	x	-	XNOR
	CB2 =	-	x	x	-	x	-	x	x	-	-	x	-	x	-	-	x	XOR
	CB3 =	x	x	x	x	x	-	-	-	x	x	x	-	-	-	-	-	XOR
	CB4 =	-	-	-	x	x	x	x	x	-	-	-	-	-	x	x	x	XOR
	CB5 =	-	-	-	-	-	-	-	-	x	x	x	x	x	x	x	x	XOR
	CB6 =	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	XOR
	CB7 =	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	XOR
DATA BITS		0 0	0 1	0 2	0 3	0 4	0 5	0 6	0 7	0 8	0 9	1 0	1 1	1 2	1 3	1 4	1 5	

16 BIT OR MASTER

	2								3								OPERATION
	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	
CB0 =	-	x	x	x	-	x	x	-	-	x	x	-	-	x	-	-	XOR
CB1 =	x	x	x	-	-	x	-	x	x	x	-	-	-	-	-	x	XOR
CB2 =	-	x	x	x	-	x	x	x	-	-	x	x	-	-	-	-	XOR
CB3 =	x	x	-	-	x	-	x	x	x	-	-	x	x	-	-	-	XOR
CB4 =	x	x	-	-	x	x	x	x	-	-	-	-	x	-	x	-	XOR
CB5 =	-	-	-	x	x	x	x	x	-	-	-	-	-	x	x	x	XOR
CB6 =	-	-	-	-	-	-	-	-	x	x	x	x	x	x	x	x	XOR
CB7 =	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	XOR
DATA BITS	1 6	1 7	1 8	1 9	2 0	2 1	2 2	2 3	2 4	2 5	2 6	2 7	2 8	2 9	3 0	3 1	

SLAVE #1

BYTE NUMBER		4								5								6								7								8								9								OPERATION
BIT NUMBER		0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	
CHECK BITS	CB0 =	x	x	-	x	-	x	x	-	x	-	-	x	-	x	-	-	x	-	x	-	x	x	-	-	x	-	x	x	-	-	x	-	-	x	x	x	-	x	x	-	-	x	-	-	x	-	-	-	XOR
	CB1 =	x	-	x	-	-	x	-	x	-	x	-	x	x	-	x	-	-	x	x	-	-	-	x	-	x	x	x	-	-	-	x	-	-	x	x	x	-	x	x	x	-	-	x	x	-	-	-	-	XOR
	CB2 =	-	x	x	-	x	-	x	x	-	-	x	-	x	-	-	x	-	x	x	x	-	x	x	-	-	x	x	-	-	x	-	-	x	-	-	x	-	x	x	-	-	x	x	-	-	x	-	x	XOR
	CB3 =	x	x	x	x	x	-	-	-	x	x	x	-	-	-	-	-	x	-	x	-	-	x	x	-	x	x	-	-	x	x	-	-	-	x	x	x	x	-	-	x	x	x	x	-	-	x	-	-	XOR
	CB4 =	-	-	-	x	x	x	x	x	-	-	-	-	-	x	x	x	-	-	-	x	x	x	x	x	-	-	-	-	-	x	x	x	-	x	x	-	-	-	x	x	x	x	x	-	-	-	x	-	XOR
	CB5 =	x	x	x	x	x	x	x	x	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	x	x	x	x	x	x	x	x	x	-	x	x	x	x	-	x	-	-	-	x	-	-	-	x	XOR
	CB6 =	x	x	x	x	x	x	x	x	-	-	-	-	-	-	-	-	x	x	x	x	x	x	x	x	-	-	-	-	-	-	-	-	x	x	-	-	x	x	x	x	-	-	-	x	-	x	-	-	XOR
	CB7 =	-	-	-	-	-	-	-	-	x	x	x	x	x	x	x	x	-	-	-	-	-	-	-	-	x	x	x	x	x	x	x	x	-	-	-	-	-	-	-	-	x	x	x	x	x	x	x	x	XOR
DATA BITS		3 2	3 3	3 4	3 5	3 6	3 7	3 8	3 9	4 0	4 1	4 2	4 3	4 4	4 5	4 6	4 7	4 8	4 9	5 0	5 1	5 2	5 3	5 4	5 5	5 6	5 7	5 8	5 9	6 0	6 1	6 2	6 3	6 4	6 5	6 6	6 7	6 8	6 9	7 0	7 1	7 2	7 3	7 4	7 5	7 6	7 7	7 8	7 9	

SLAVE #2 SLAVE #3 SLAVE #4

Table 6. 8206 Syndrome Decoding

Syndrome Bits 7	6	5	4	0 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 2 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 3 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
0	0	0	0	N CB0 CB1 D CB2 D D 18 CB3 D D 0 D 1 2 D
0	0	0	1	CB4 D D 5 D 6 7 D D 3 16 D 4 D D 17
0	0	1	0	CB5 D D 11 D 19 12 D D 8 9 D 10 D D 67
0	0	1	1	D 13 14 D 15 D D 21 20 D D 66 D 22 23 D
0	1	0	0	CB6 D D 25 D 26 49 D D 48 24 D 27 D D 50
0	1	0	1	D 52 55 D 51 D D 70 28 D D 65 D 53 54 D
0	1	1	0	D 29 31 D 64 D D 69 68 D D 32 D 33 34 D
0	1	1	1	30 D D 37 D 38 39 D D 35 71 D 36 D D U
1	0	0	0	CB7 D D 43 D 77 44 D D 40 41 D 42 D D U
1	0	0	1	D 45 46 D 47 D D 74 72 D D U D 73 U D
1	0	1	0	D 59 75 D 79 D D 58 60 D D 56 D U 57 D
1	0	1	1	63 D D 62 D U U D D U U D 61 D D U
1	1	0	0	D U U D U D D U 76 D D U D U U D
1	1	0	1	78 D D U D U U D D U U D U D D U
1	1	1	0	U D D U D U U D D U U D U D D U
1	1	1	1	D U U D U D D U U D D U D U U D

N = No Error
CBX = Error in Check Bit X
X = Error in Data Bit X
D = Double Bit Error
U = Uncorrectable Multi-Bit Error

SYSTEM ENVIRONMENT

The 8206 interface to a typical 32 bit memory system is illustrated in Figure 5. For larger systems, the partial parity bits from slaves two to four must be

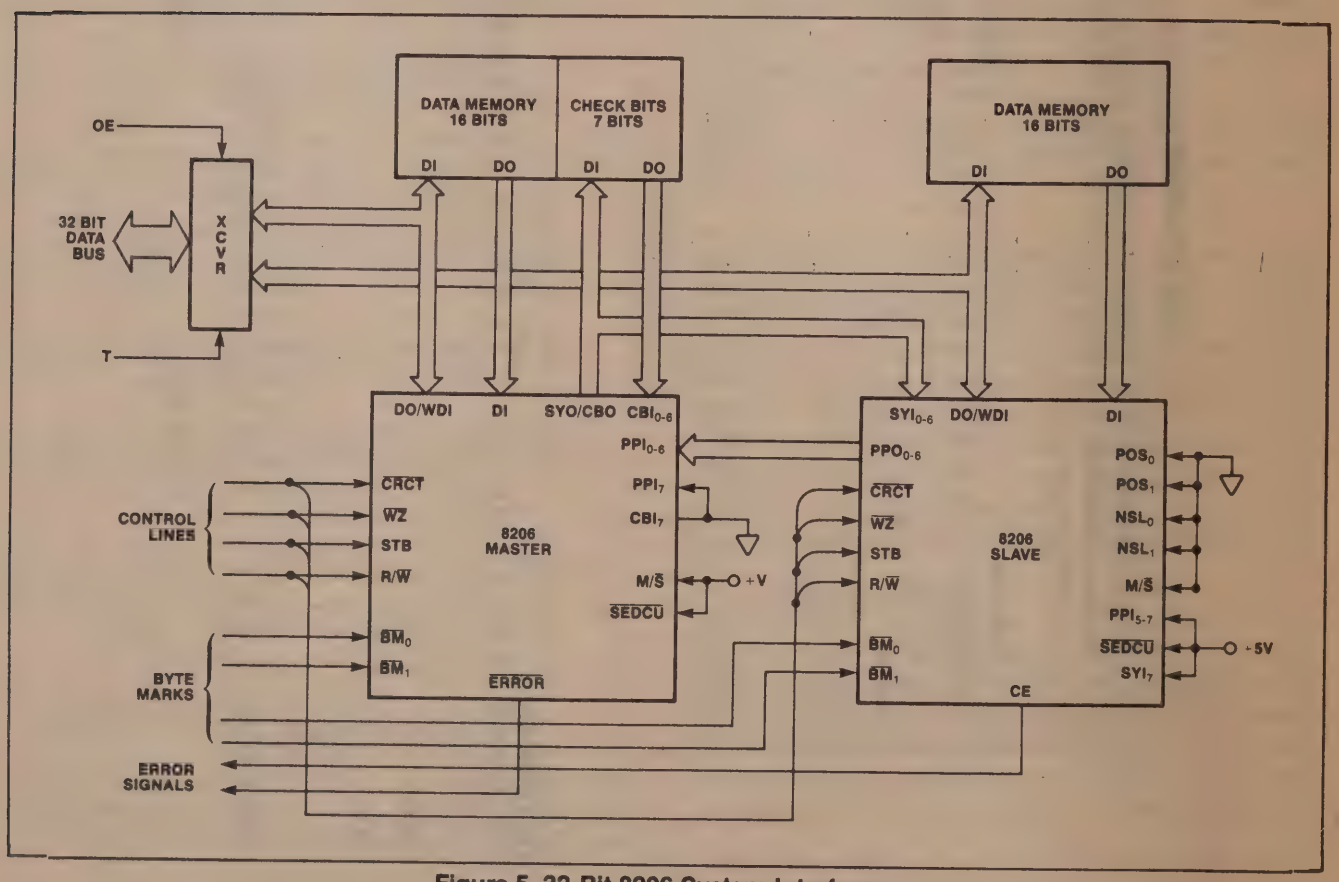


Figure 5. 32-Bit 8206 System Interface

205220-007

XOR'ed externally, which calls for one level of XOR gating for three 8206's and two levels for four or five 8206's.

The 8206 is designed for direct connection to the Intel 8207 Advanced Dynamic RAM Controller. The 8207 has the ability to perform dual port memory control, and Figure 6 illustrates a highly integrated dual port RAM implementation using the 8206 and 8207. The 8206/8207 combination permits such features as automatic scrubbing (correcting errors in memory during refresh), extending RAS and CAS timings for Read-Modify-Writes in single memory cycles, and automatic memory initialization upon reset. Together these two chips provide a complete dual-port, error-corrected dynamic RAM subsystem.

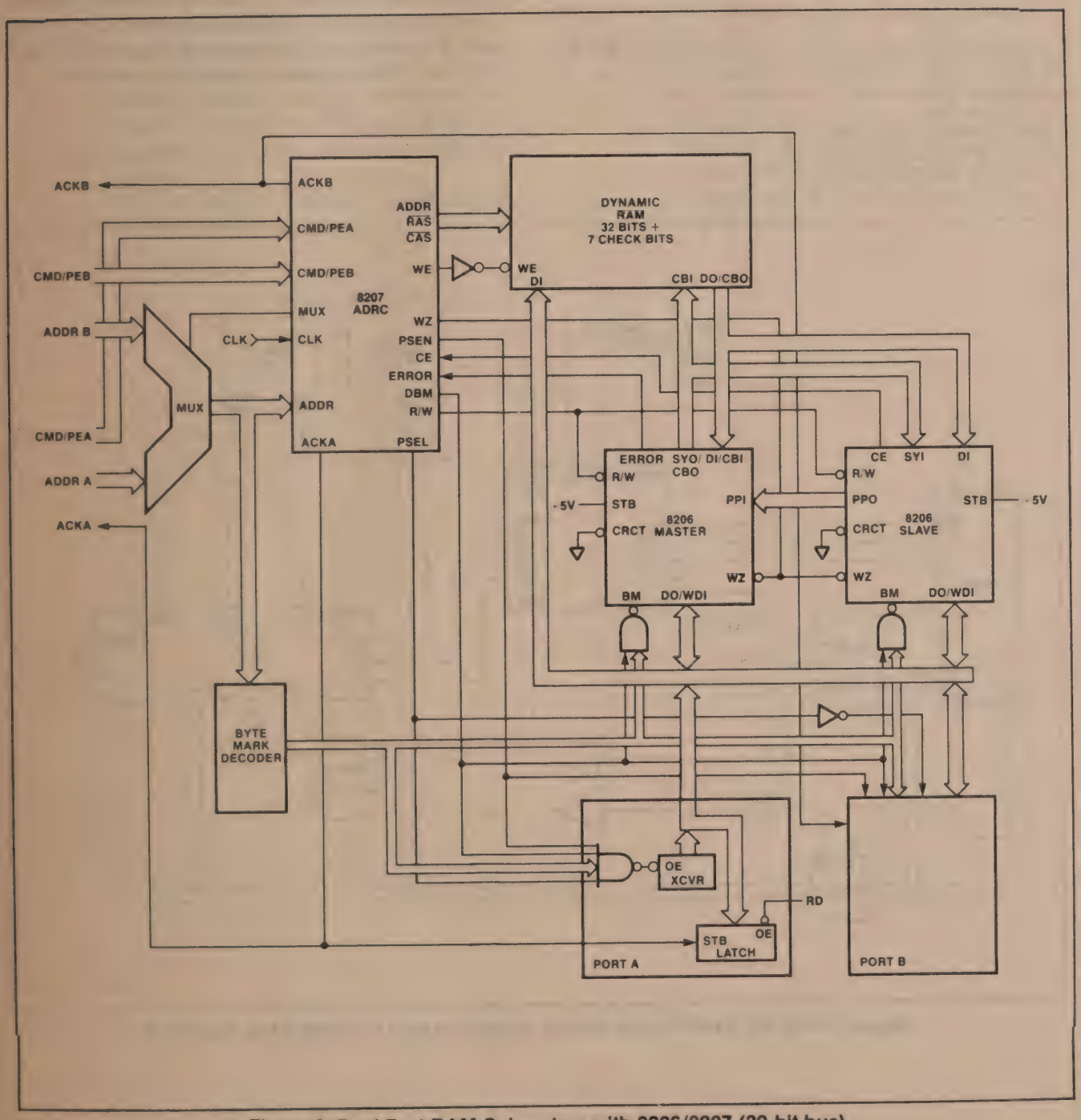


Figure 6. Dual Port RAM Subsystem with 8206/8207 (32-bit bus)

Table 7. 8206-2 Syndrome Decoding

Syndrome Bits 5	4	3	0 1 2	0 0 0	1 0 0	0 1 0	1 1 0	0 0 1	1 0 1	0 1 1	1 1 1
0	0	0		N	CB0	CB1	D	CB2	D	D	D
0	0	1		CB3	D	D	0	D	1	2	D
0	1	0		CB4	D	D	5	D	6	7	D
0	1	1		D	3	D	D	4	D	D	D
1	0	0		CB5	D	D	11	D	D	12	D
1	0	1		D	8	9	D	10	D	D	D
1	1	0		D	13	14	D	15	D	D	D
1	1	1		D	D	D	D	D	D	D	D

N = No Error
CBX = Error in Check Bit X
X = Error in Data Bit X
D = Double Bit Error

The 8206-2 handles 8 or 16 bits of data. For 8 bit 8206-2 systems, the DI_{8-15}, DO/WDI_{8-15} and $\overline{BM}_1$ inputs are grounded.

The 8206-2 is designed for direct connection to the Intel 8207-2 Advanced Dynamic RAM Controller. The 8207-2 has the ability to perform dual port memory control, and Figure 7 illustrates a highly integrated iAPX 186 RAM implementation using the 8206-2 and 8207-2. The 8206-2/8207-2 combination permits such features as automatic scrubbing (correcting errors in memory during refresh), extending RAS and CAS timings for Read-Modify-Writes in single memory cycles, and automatic memory initialization upon reset. Together these two chips provide a complete dual-port, error-corrected dynamic RAM subsystems.

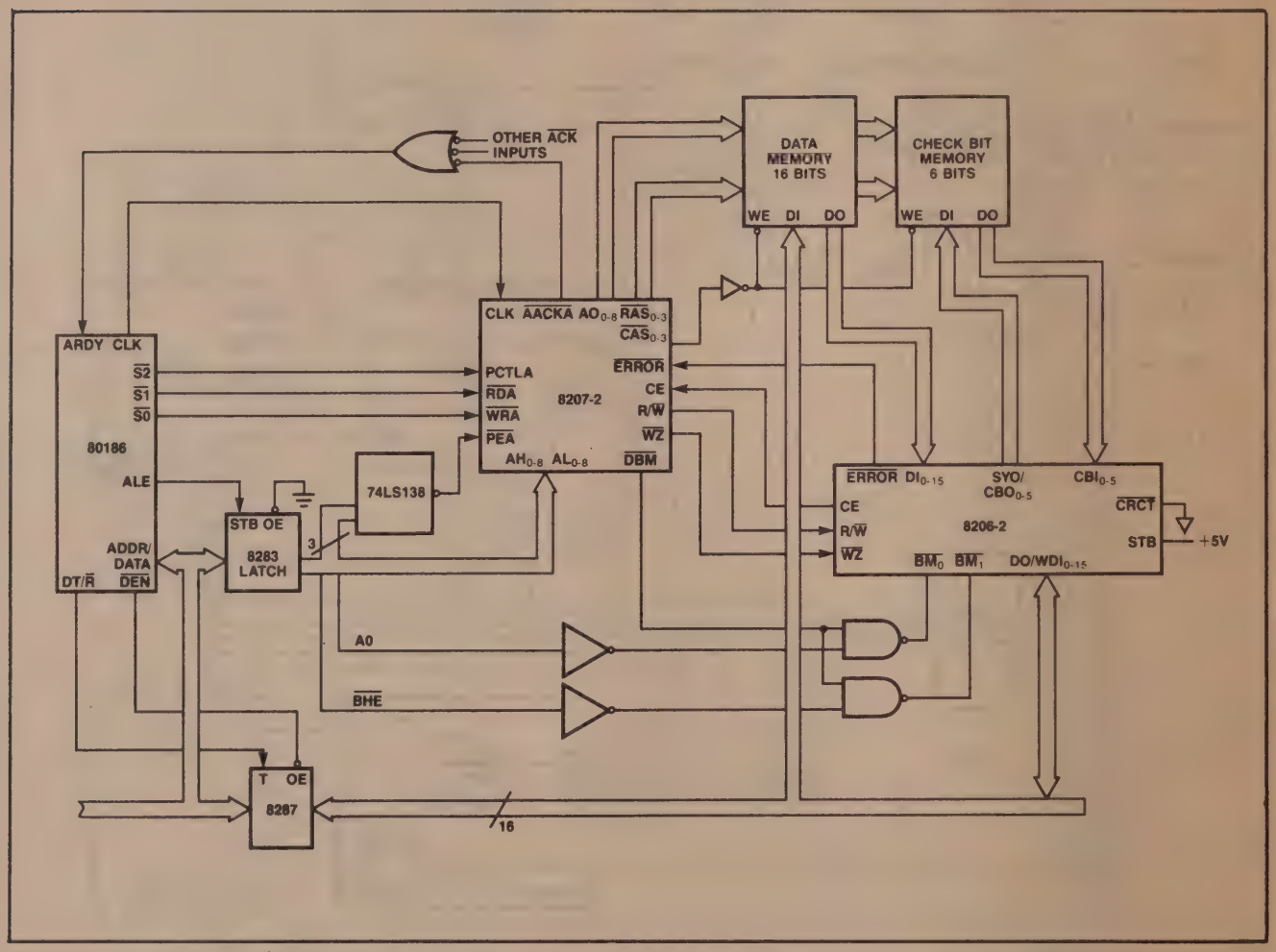


Figure 7. iAPX 186 RAM Correct Always Subsystem with the 8206-2 and the 8207-2

205220-007

MEMORY BOARD TESTING

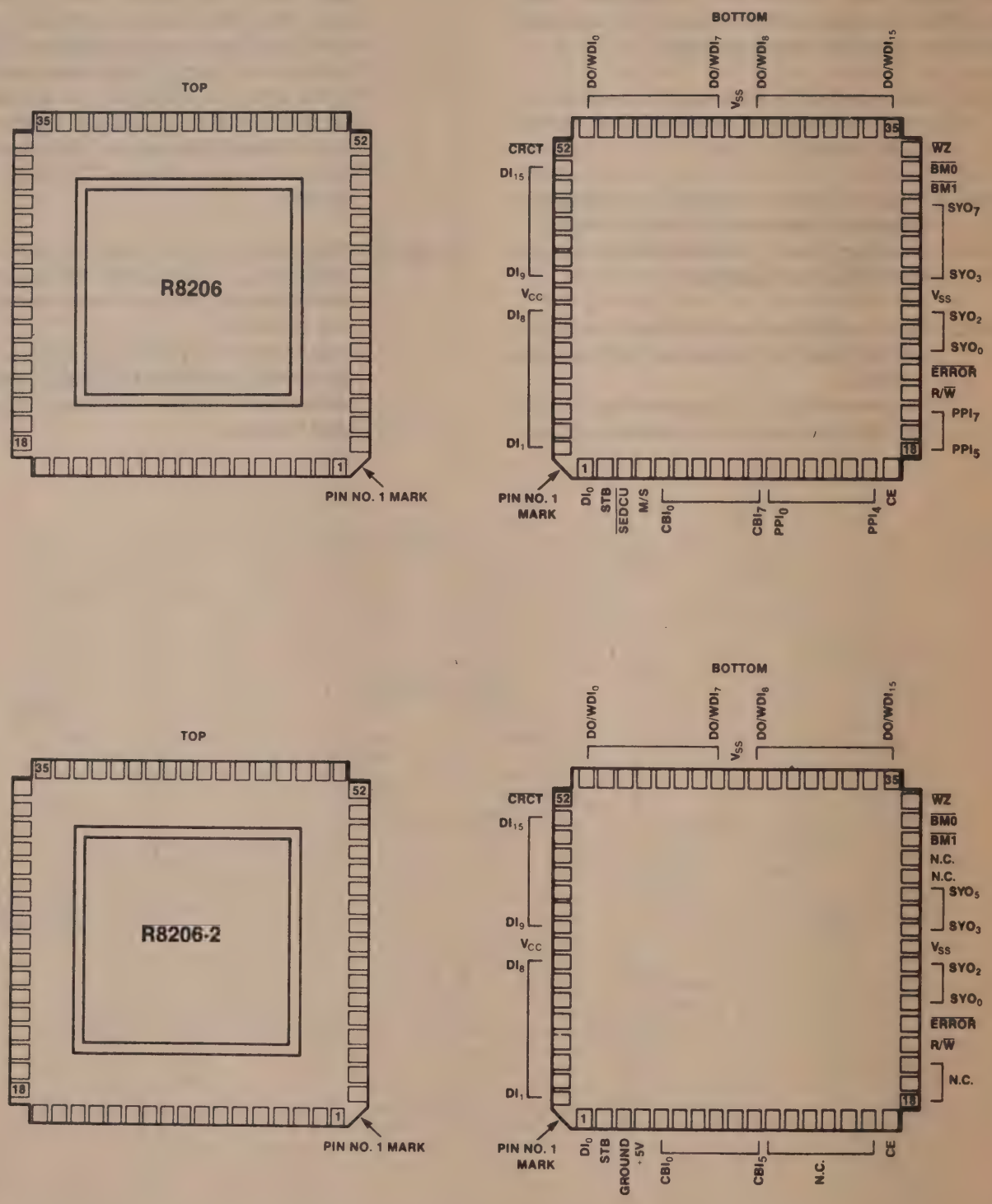
The 8206 lends itself to straightforward memory board testing with a minimum of hardware overhead. The following is a description of four common test modes and their implementation.

Mode 0—Read and write with error correction.
Implementation: This mode is the normal 8206 operating mode.

Mode 1—Read and write data with error correction disabled to allow test of data memory.
Implementation: This mode is performed with $\overline{CRCT}$ deactivated.

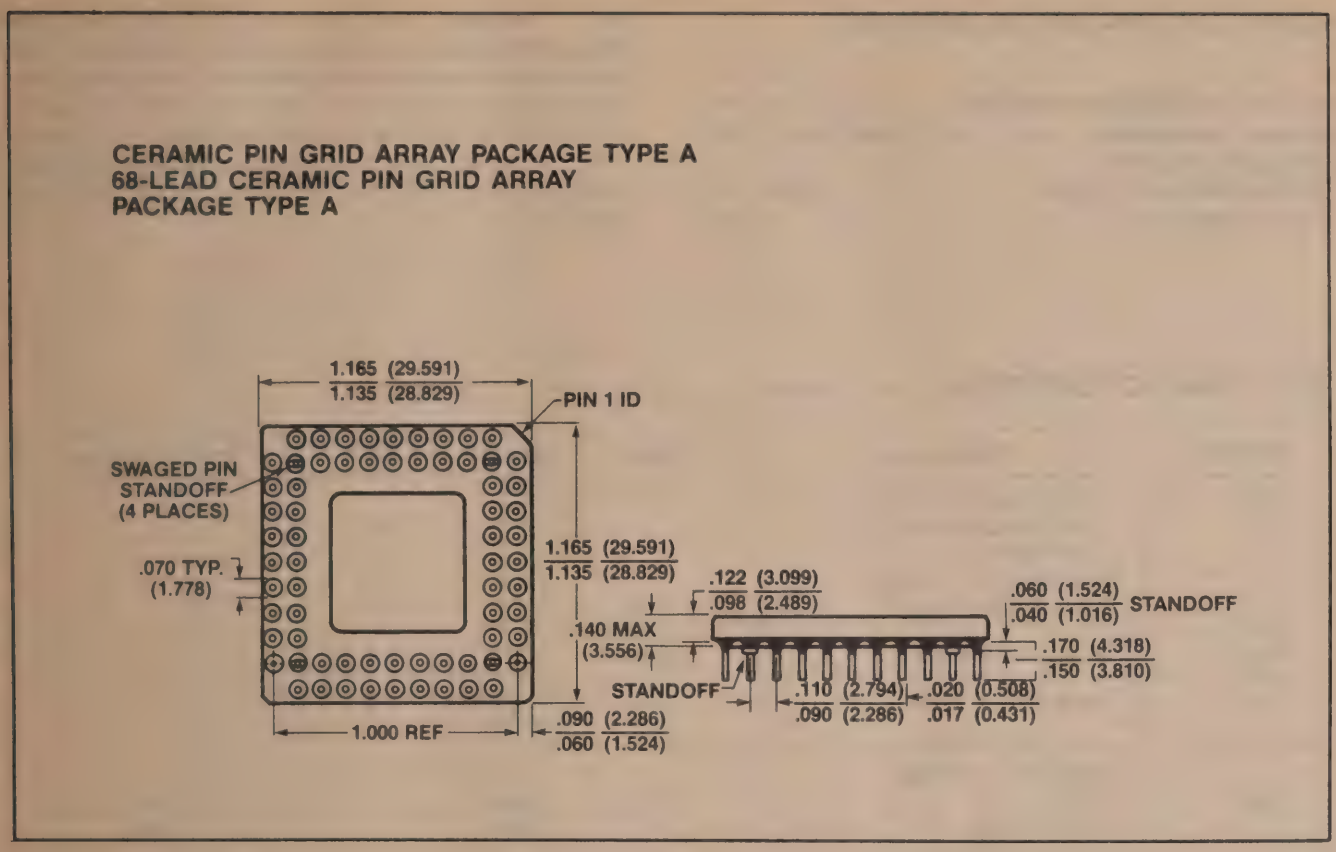
Mode 2—Read and write check bits with error correction disabled to allow test of check bits memory.
Implementation: Any pattern may be written into the check bits memory by judiciously choosing the proper data word to generate the desired check bits, through the use of the 8206 Hamming code. To read out the check bits it is first necessary to fill the data memory with all zeros, which may be done by activating $\overline{WZ}$ and incrementing memory addresses with $\overline{WE}$ to the check bits memory held inactive, and then performing ordinary reads. The check bits will then appear directly at the SYO outputs, with bits CB0 and CB1 inverted.

Mode 3—Write data, without altering or writing check bits, to allow the storage of bit combinations to cause error correction and detection.
Implementation: This mode is implemented by writing the desired word to memory with $\overline{WE}$ to the check bits array held inactive.



NOTE:
The 8206 and 8206-2 is packaged in a 68 pin JEDEC TYPE A hermetic chip carrier

Figure 8a. 8206 and 8206-2 Pinout Diagram



PIN GRID ARRAY (PGA) PIN-OUT
TOP VIEW

• 68 • 66 • 64 • 62 • 60 • 58 • 56 • 54 • 52
• 1 • 2 • 67 • 65 • 63 • 61 • 59 • 57 • 55 • 53 • 51
• 3 • 4 • 50 • 49
• 5 • 6 • 48 • 47
• 7 • 8 • 46 • 45
• 9 • 10 • 44 • 43
• 11 • 12 • 42 • 41
• 13 • 14 • 40 • 39
• 15 • 16 • 38 • 37
• 17 • 19 • 21 • 23 • 25 • 27 • 29 • 31 • 33 • 36 • 35
• 18 • 20 • 22 • 24 • 26 • 28 • 30 • 32 • 34

Figure 8b. 8206 Pin Grid Array Package and Pinout Diagram

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias ..... 0°C to 70°C
Storage Temperature ..... −65°C to +150°C
Voltage On Any Pin
With Respect to Ground ..... −0.5V to +7V
Power Dissipation ..... 1.5 Watts

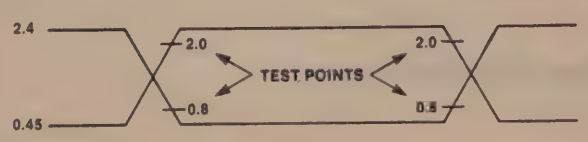
**NOTE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = 5.0V ± 10%, V_{SS} = GND)

Symbol	Parameter	Min.	Max.	Unit	Test Conditions
I_{CC}	Power Supply Current —Single 8206, 8206-2 or Slave #1 —Master in Multi-Chip or Slaves #2, 3, 4		270 230	mA mA	
V_{IL}[1]	Input Low Voltage	−0.5	0.8	V	
V_{IH}[1]	Input High Voltage	2.0	V_{CC} + 0.5V	V	
V_{OL}	Output Low Voltage —DO —All Others		0.45 0.45	V V	I_{OL} = 8mA I_{OL} = 2.0mA
V_{OH}	Output High Voltage —DO, CBO —All Other Outputs	2.6 2.4		V V	I_{OH} = −2mA I_{OH} = −0.4mA
I_{LO}	I/O Leakage Current —PPI_4/CE —DO/WDI_{0-15}		± 20 ± 10	μA μA	$0.45V \leq V_{I/O} \leq V_{CC}$
I_{LI}	Input Leakage Current —$PPI_{0-3,\ 5-7}$, CBI_{6-7}, $\overline{SEDCU}$[2] —All Other Input Only Pins		± 20 ± 10	μA μA	$0V \leq V_{IN} \leq V_{CC}$

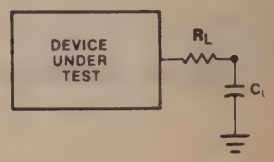
NOTES:
1. $\overline{SEDCU}$ (pin 3) and M/$\overline{S}$ (pin 4) are device strapping options and should be tied to V_{CC} or GND. V_{IH} min = V_{CC} −0.5V and V_{IL} max = 0.5V.
2. PPI_{0-7} (pins 13–20) and CBI_{6-7} (pins 11, 12) have internal pull-up resistors and if left unconnected will be pulled to V_{CC}.

A.C. TESTING INPUT, OUTPUT WAVEFORM



A.C. TESTING: INPUTS ARE DRIVEN AT 2.4V FOR A LOGIC "1" AND 0.45V FOR A LOGIC "0." TIMING MEASUREMENTS ARE MADE AT 2.0V FOR A LOGIC "1" AND 0.8V FOR A LOGIC "0."

A.C. TESTING LOAD CIRCUIT



C_L INCLUDES JIG CAPACITANCE

205220-007

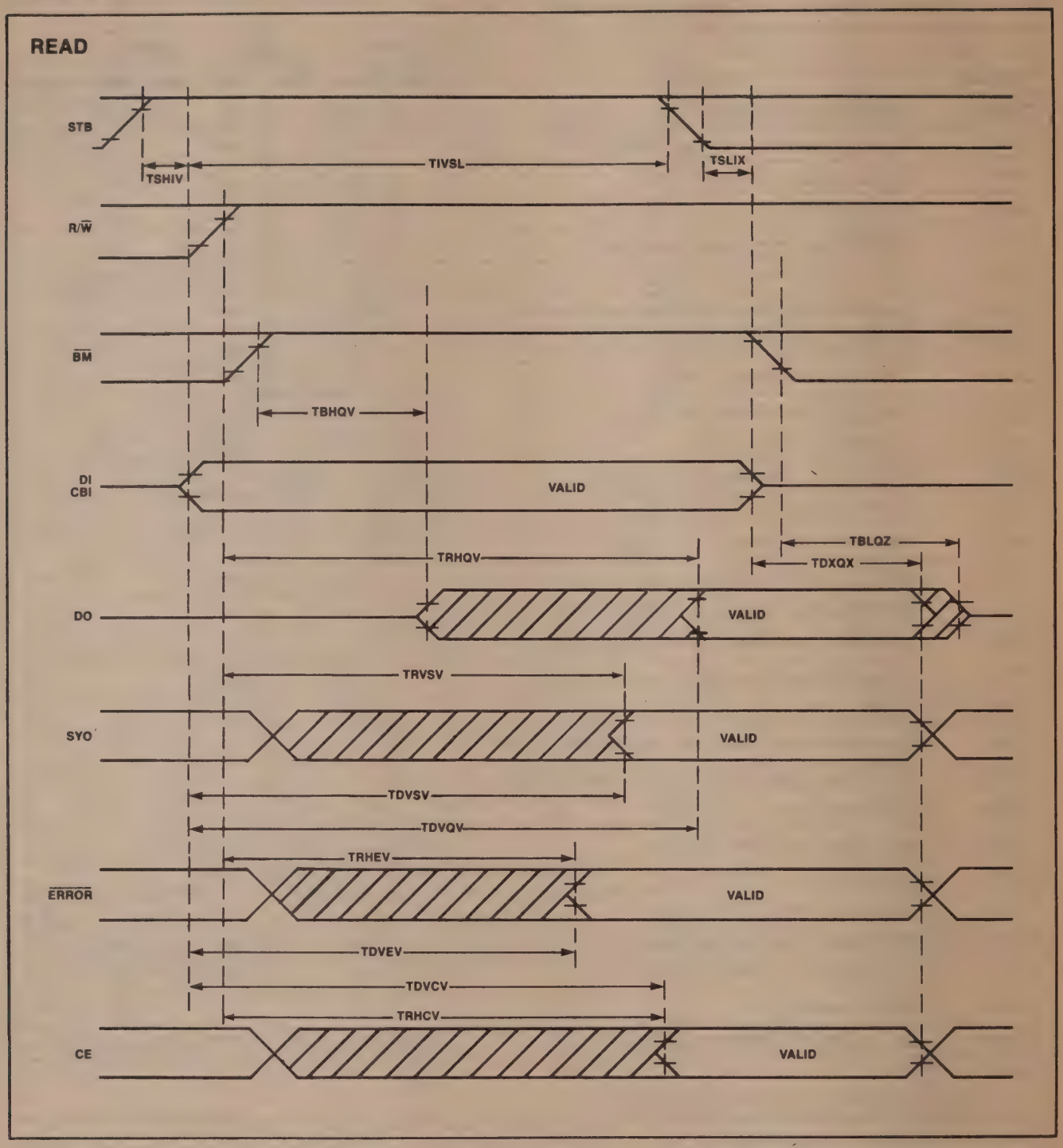
A.C. CHARACTERISTICS (T_A = 0°C to 70°C. V_{CC} = +5V ± 10%, V_{SS} = 0V, R_L = 22Ω, C_L = 50 pF; all times are in nsec.)

Symbol	Parameter	8206-1 Min.	8206-1 Max.	8206 Min.	8206 Max.	8206-2 Min.	8206-2 Max.	Notes
TRHEV	$\overline{ERROR}$ Valid from R/$\overline{W}$↑		20		25		40	4
TRHCV	CE Valid from R/$\overline{W}$↑ (Single 8206)		34		44		49	
TRHQV	Corrected Data Valid from R/$\overline{W}$↑		44		54		66	1
TRVSV	SYO/CBO/PPO Valid from R/$\overline{W}$		32		42		46	1
TDVEV	$\overline{ERROR}$ Valid from Data/Check Bits In		35		42		57	
TDVCV	CE Valid from Data/Check Bits In		50		70		76	
TDVQV	Corrected Data Valid from Data/Check Bits In		55		67		74	
TDVSV	SYO/PPO Valid from Data/Check Bits In		40		55		65	
TBHQV	Corrected Data Access Time		35		37		37	
TBXQX	Hold Time From Data/Check Bits In	0		0		0		1
TBLQZ	Corrected Data Float Delay	0	25	0	28	0	28	1
TSHIV	STB High to Data/Check Bits In Valid	30		30		30		2
TIVSL	Data/Check Bits In to STB↓ Set-up	5		5		5		
TSLIX	Data/Check Bits In from STB↓ Hold	15		25		25		
TPVEV	$\overline{ERROR}$ Valid from Partial Parity In		21		30			3,4
TPVQV	Corrected Data (Master) from Partial Parity In		46		61			1,3
TPVSV	Syndrome/Check Bits Out from Partial Parity In		32		43			1,3
TSVQV	Corrected Data (Slave) Valid from Syndrome		41		51			3
TSVCV	CE Valid from Syndrome (Slave number 1)		43		48			3
TQVQV	Check Bits/Partial Parity Out from Write Data In		44		64		69	1
TRHSX	Check Bits/Partial Parity Out from R/W,$\overline{WZ}$ Hold	0		0		0		1
TRLSX	Syndrome Out from R/$\overline{W}$ Hold	0		0		0		
TQXQX	Hold Time from Write Data In	0		0		0		1
TSVRL	Syndrome Out to R/$\overline{W}$↓ Set-up	5		17				3
TDVRL	Data/Check Bits to R/$\overline{W}$ Set-up	24		39		41		1
TDVQU	Uncorrected Data Out from Data In		29		32		38	
TTVQV	Corrected Data Out from $\overline{CRCT}$↓		25		30		33	
TWLQL	$\overline{WZ}$↓ to Zero Out		25		30		34	4
TWHQX	Zero Out from $\overline{WZ}$↑ Hold	0		0		0		

NOTES:
1. A.C. Test Levels for CBO and DO are 2.4V and 0.8V.
2. T_{SHIV} is required to guarantee output delay timings: T_{DVEV}, T_{DVCV}, T_{DVQV}, T_{DVSV}. T_{SHIV} + T_{IVSL} guarantees a min STB pulse width of 35 ns (45 ns for the 8206-8).
3. Not required for 8/16 bit systems
4. 8206 S40037 has three parameters relaxed from full spec 8206: T_{RHEV} = 35ns, T_{PVEV} = 40ns, T_{WLQL} = 40ns.

WAVEFORMS

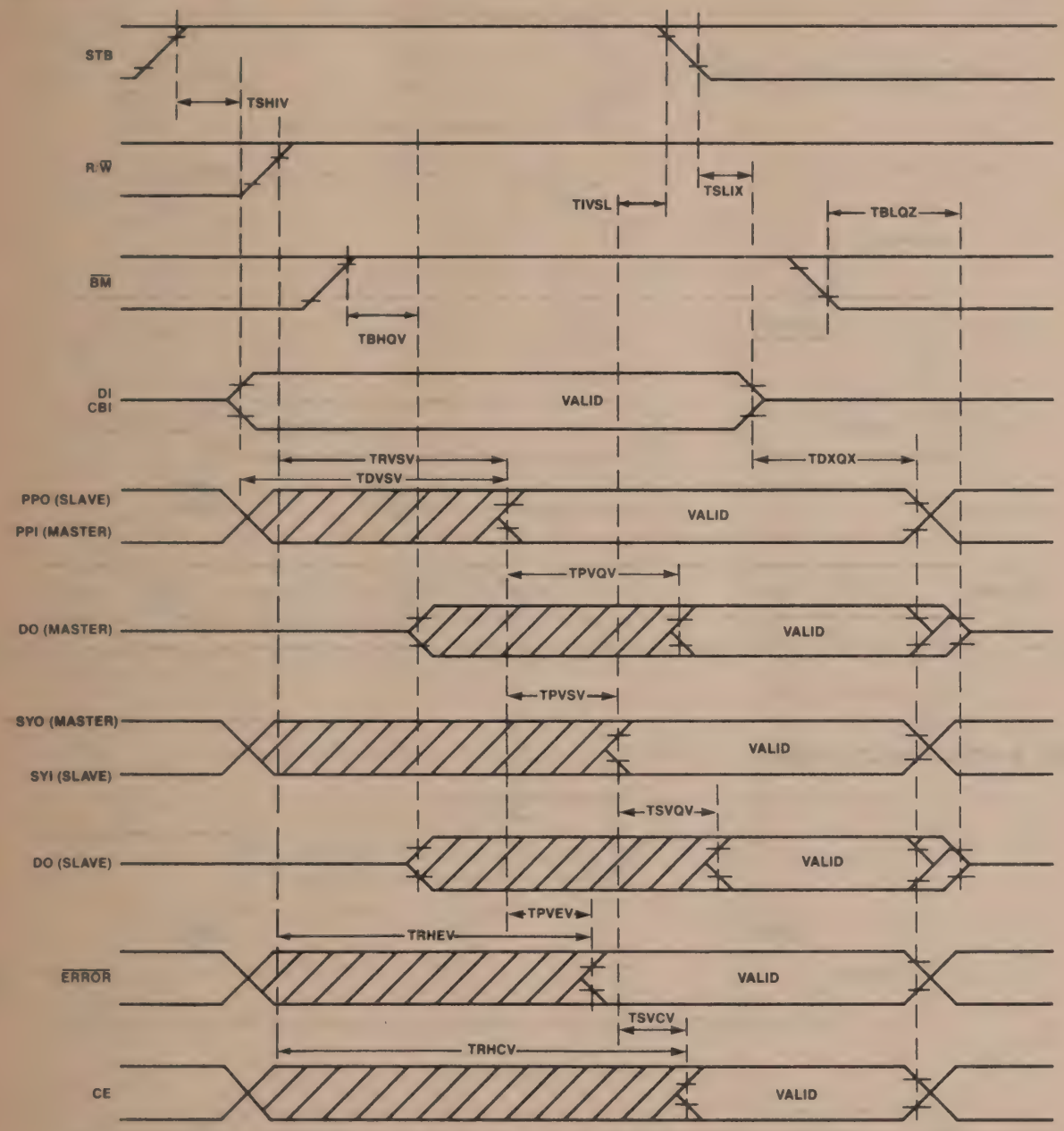
READ



205220-007

WAVEFORMS (Continued)

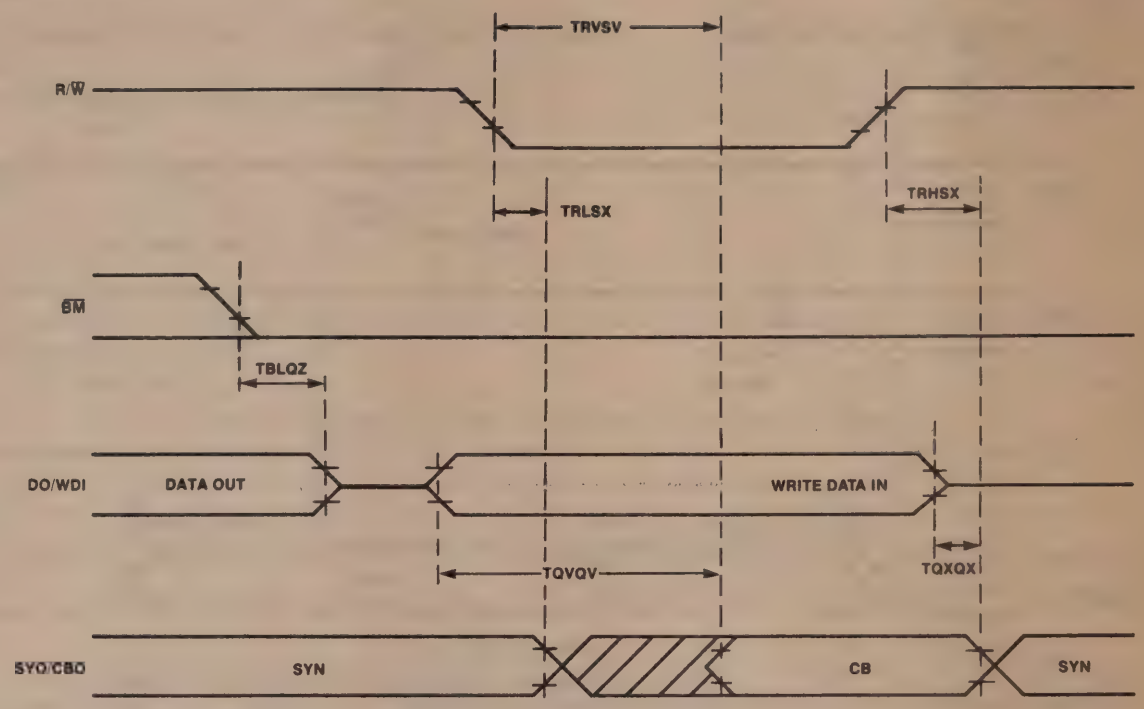
READ—MASTER/SLAVE



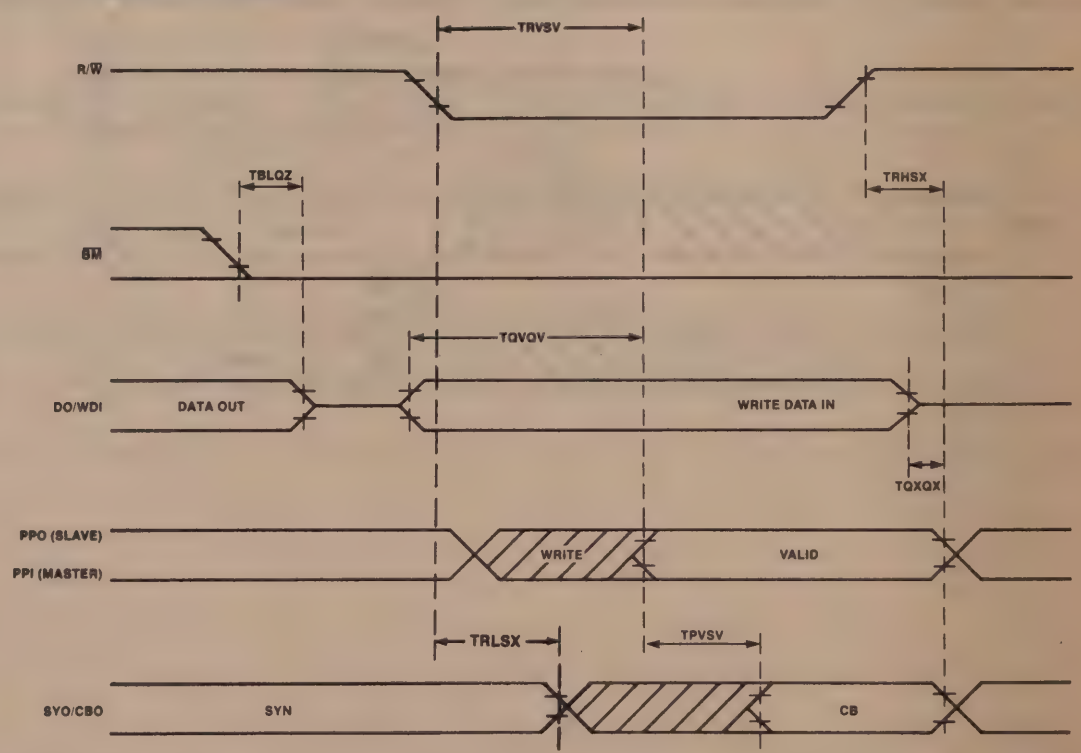
205220-007

WAVEFORMS (Continued)

FULL WRITE

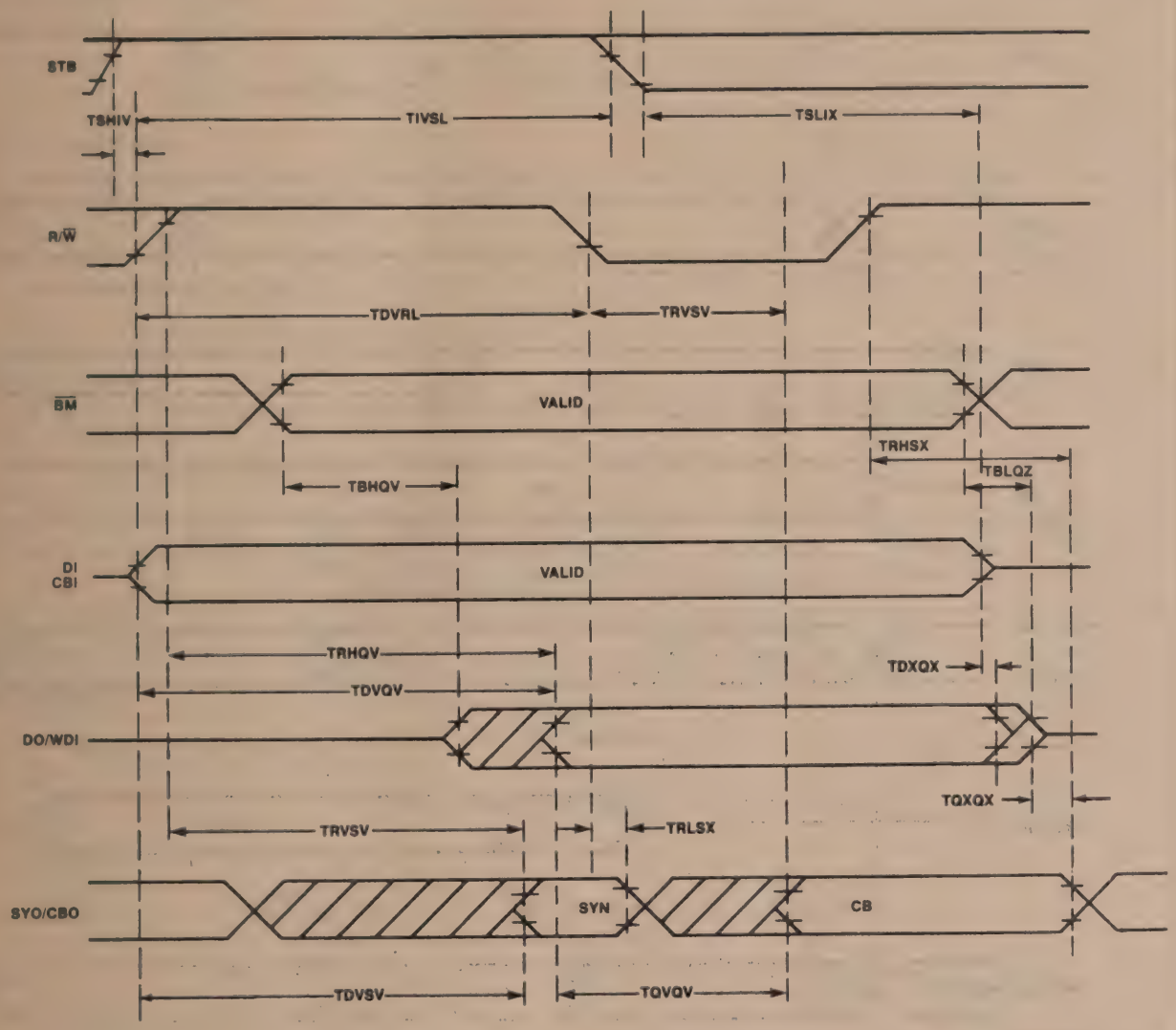


FULL WRITE—MASTER/SLAVE



WAVEFORMS (Continued)

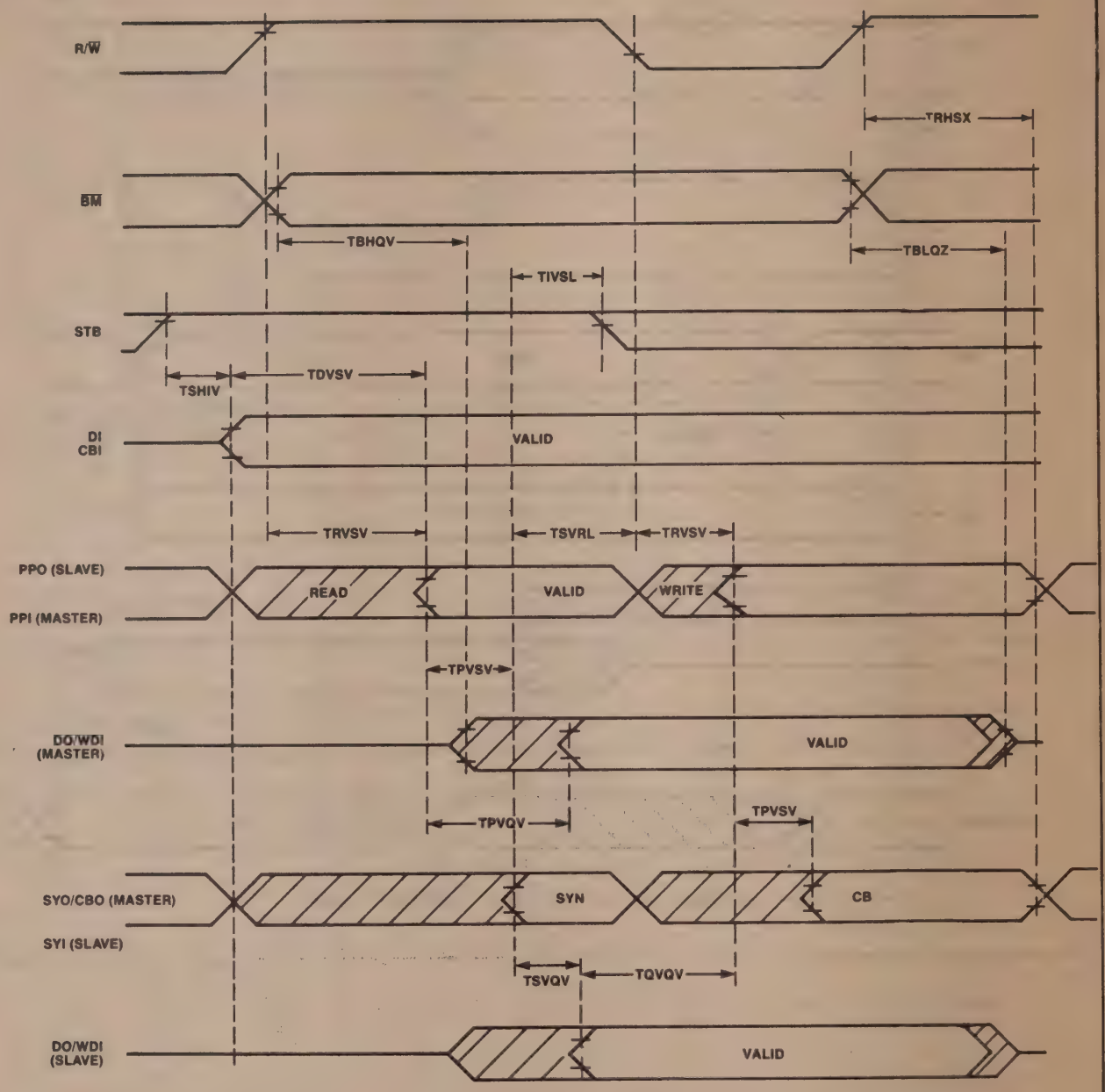
READ MODIFY WRITE



205220-007

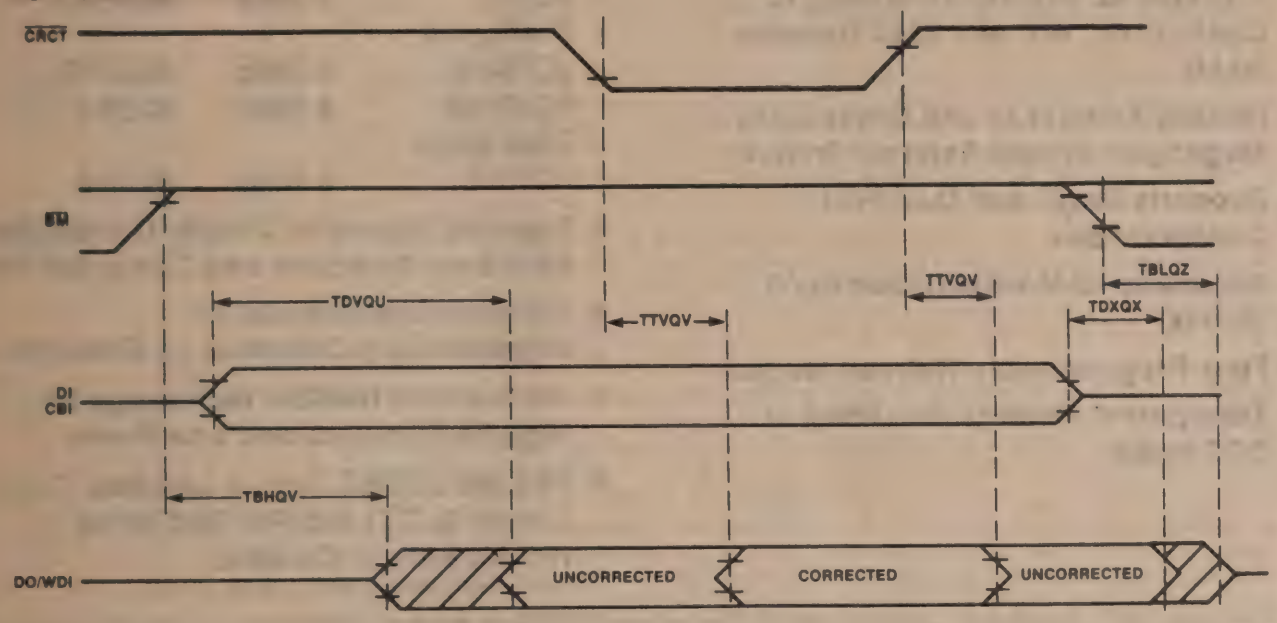
WAVEFORMS (Continued)

READ MODIFY WRITE—MASTER/SLAVE

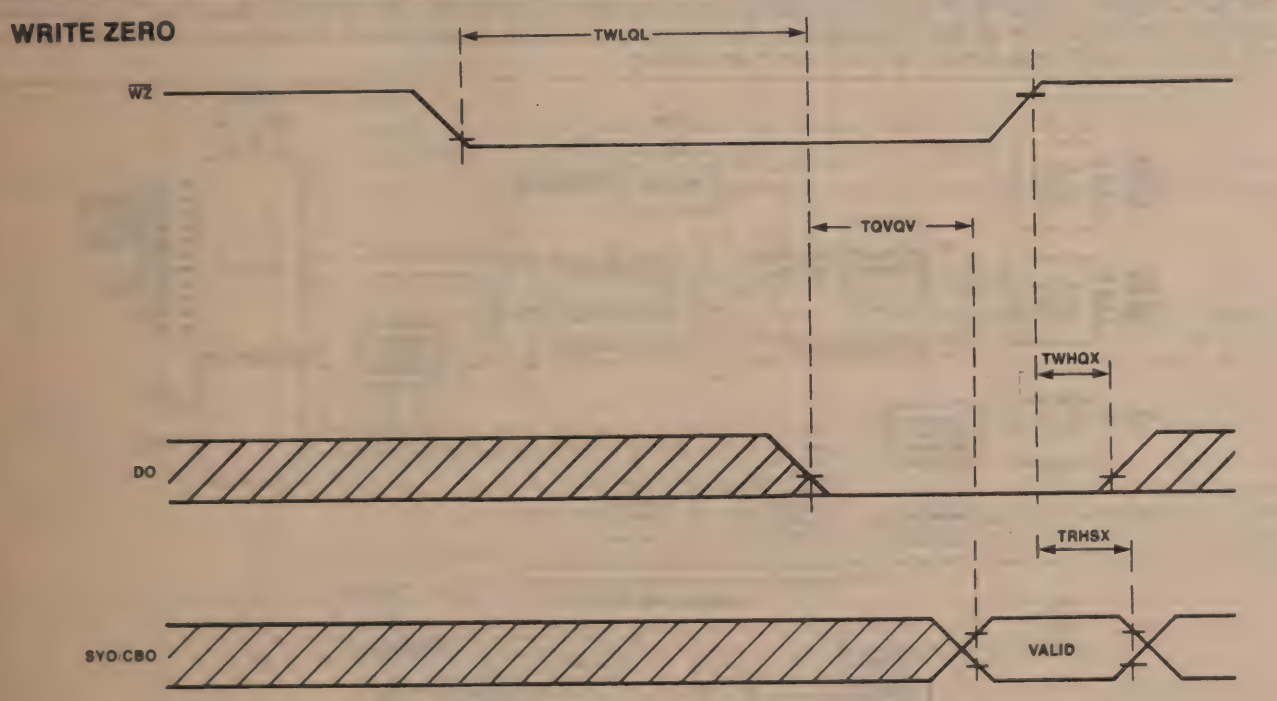


WAVEFORMS (Continued)

NON-CORRECTING READ



WRITE ZERO



intel

PRELIMINARY

8207
DUAL-PORT DYNAMIC RAM CONTROLLER

- **Provides All Signals Necessary to Control 16K, 64K and 256K Dynamic RAMs**
- **Directly Addresses and Drives up to 2 Megabytes without External Drivers**
- **Supports Single and Dual-Port Configurations**
- **Automatic RAM Initialization in All Modes**
- **Four Programmable Refresh Modes**
- **Transparent Memory Scrubbing in ECC Mode**
- **80286 Fast cycle (CFS=1) — 8 MHz — 8207-16; 6 MHz — 8207-12**
  8086/186 Slow cycle (CFS=0) — 8 MHz — 8207-8; 6 MHz — 8207-6
- **Provides Signals to Directly Control the 8206 Error Detection and Correction Unit**
- **Supports Synchronous or Asynchronous Operation on Either Port**
- **+5 Volt Only HMOSII Technology for High Performance and Low Power**
- **68 Lead JEDEC Type A Leadless Chip Carrier (LCC) and Pin Grid Array (PGA), Both in Ceramic.**
  See Packaging Specifications, Order #: 231369

The Intel 8207 Advanced Dynamic RAM Controller (ADRC) is a high-performance, systems-oriented, Dynamic RAM controller that is designed to easily interface 16K, 64K and 256K Dynamic RAMs to Intel and other microprocessor systems. A dual-port interface allows two different busses to independently access memory. When configured with an 8206 Error Detection and Correction Unit the 8207 supplies the necessary logic for designing large error-corrected memory arrays. This combination provides automatic memory initialization and transparent memory error scrubbing.

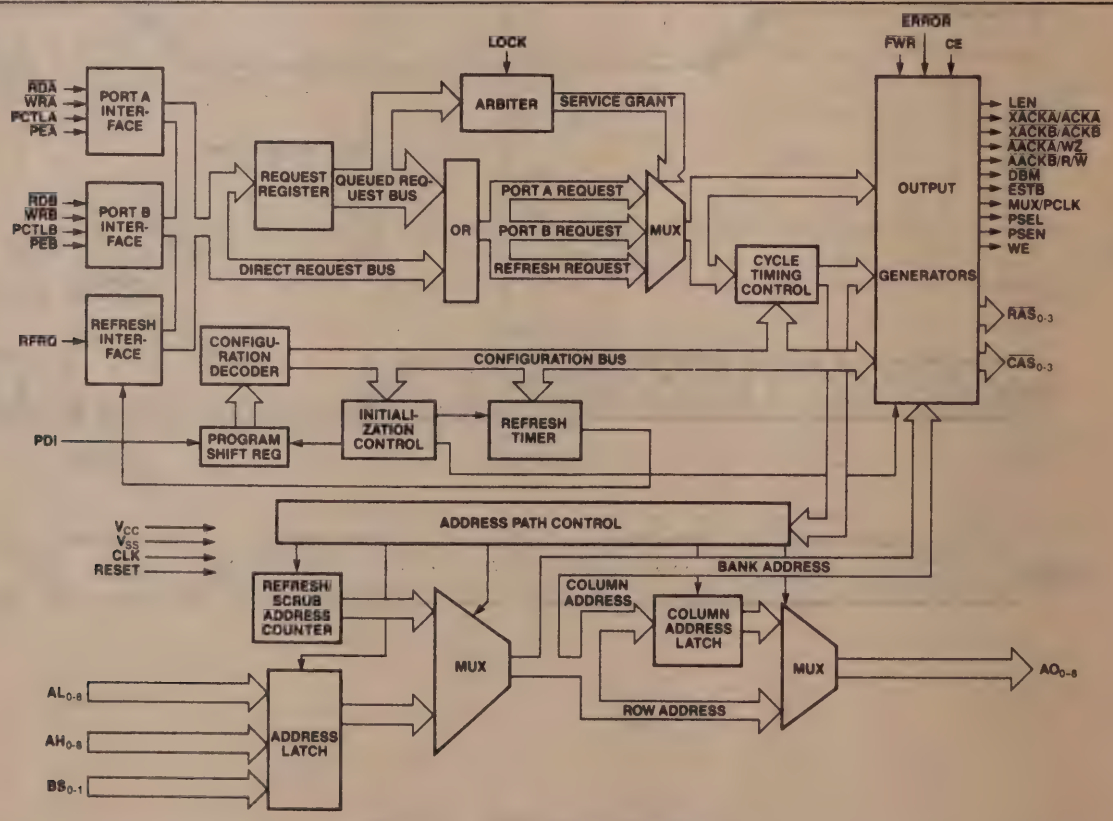


Figure 1. 8207 Block Diagram

Intel Corporation Assumes No Responsibility for the Use of Any Circuitry Other Than Circuitry Embodied in an Intel Product. No Other Circuit Patent Licenses are Implied. Information Contained Herein Supercedes Previously Published Specifications On These Devices From Intel.
©INTEL CORPORATION, 1983

JAN. 1986
ORDER NUMBER: 210463-005

Table 1. Pin description

Symbol	Pin	Type	Name and Function
LEN	1	O	**ADDRESS LATCH ENABLE:** In two-port configurations, when Port A is running with iAPX 286 Status interface mode, this output replaces the ALE signal from the system bus controller of port A and generates an address latch enable signal which provides optimum setup and hold timing for the 8207. This signal is used in Fast Cycle operation only.
$\overline{XACKA}$/ $\overline{ACKA}$	2	O	**TRANSFER ACKNOWLEDGE PORTA/ACKNOWLEDGE PORTA:** In non-ECC mode, this pin is $\overline{XACKA}$ and inidcates that data on the bus is valid during a read cycle or that data may be removed from the bus during a write cycle for Port A. $\overline{XACKA}$ is a Multibus-compatible signal. In ECC mode, this pin is $\overline{ACKA}$ which can be configured, depending on the programming of the X program bit, as an $\overline{XACK}$ or $\overline{AACK}$ strobe. The SA programming bit determines whether the $\overline{AACK}$ will be an early $\overline{EAACKA}$ or a late $\overline{LAACKA}$ interface signal.
$\overline{XACKB}$/ $\overline{ACKB}$	3	O	**TRANSFER ACKNOWLEDGE PORT B/ACKNOWLEDGE PORT B:** In non-ECC mode, this pin is XACKB and indicates that data on the bus is valid during a read cycle or that data may be removed from the bus during a write cycle for Port B. $\overline{XACKB}$ is a Multibus-compatible signal. In ECC mode, this pin is $\overline{ACKB}$ which can be configured, depending on the programming of the X program bit, as an $\overline{XACK}$ or $\overline{AACK}$ strobe. The SB programming bit determines whether the $\overline{AACK}$ will be an early $\overline{EAACKB}$ or a late $\overline{LAACKB}$ interface signal.
$\overline{AACKA}$/ $\overline{WZ}$	4	O	**ADVANCED ACKNOWLEDGE PORT A/WRITE ZERO:** In non-ECC mode, this pin is $\overline{AACKA}$ and indicates that the processor may continue processing and that data will be available when required. This signal is optimized for the system by programming the SA program bit for synchronous or asynchronous operation. In ECC mode, after a RESET, this signal will cause the 8206 to force the data to all zeros and generate the appropriate check bits.
$\overline{AACKB}$/ R/$\overline{W}$	5	O	**ADVANCED ACKNOWLEDGE PORT B/READ/WRITE:** In non-ECC mode, this pin is $\overline{AACKB}$ and indicates that the processor may continue processing and that data will be available when required. This signal is optimized for the system by programming the SB program bit for synchronous or asynchronous operation. In ECC mode, this signal causes the 8206 EDCU to latch the syndrome and error flags and generate check bits.
$\overline{DBM}$	6	O	**DISABLE BYTE MARKS:** This is an ECC control output signal indicating that a read or refresh cycle is occurring. This output forces the byte address decoding logic to enable all 8206 data output buffers. In ECC mode, this output is also asserted during memory initialization and the 8-cycle dynamic RAM wake-up exercise. In non-ECC systems this signal indicates that either a read, refresh or 8-cycle warm-up is in progress.
$\overline{ESTB}$	7	O	**ERROR STROBE:** In ECC mode, this strobe is activated when an error is detected and allows a negative-edge triggered flip-flop to latch the status of the 8206 EDCU CE for systems with error logging capabilities. $\overline{ESTB}$ will not be issued during refresh cycles.
LOCK	8	I	**LOCK:** This input instructs the 8207 to lock out the port not being serviced at the time LOCK was issued.
V_{CC}	9 43	I	**DRIVER POWER:** +5 Volts. Supplies V_{CC} for the output drivers. **LOGIC POWER:** +5 Volts. Supplies V_{CC} for the internal logic circuits.
CE	10	I	**CORRECTABLE ERROR:** This is an ECC input from the 8206 EDCU which instructs the 8207 whether a detected error is correctable or not. A high input indicates a correctable error. A low input inhibits the 8207 from activating WE to write the data back into RAM. This should be connected to the CE output of the 8206.
$\overline{ERROR}$	11	I	**ERROR:** This is an ECC input from the 8206 EDCU and instructs the 8207 that an error was detected. This pin should be connected to the ERROR output of the 8206.
MUX/ PCLK	12	O	**MULTIPLEXER CONTROL/PROGRAMMING CLOCK:** Immediately after a RESET this pin is used to clock serial programming data into the PDI pin. In normal two-port operation, this pin is used to select memory addresses from the appropriate port. When this signal is high, port A is selected and when it is low, port B is selected. This signal may change state before the completion of a RAM cycle, but the RAM address hold time is satisfied.
PSEL	13	O	**PORT SELECT:** This signal is used to select the appropriate port for data transfer. When this signal is high port A is selected and when it is low port B is selected.
PSEN	14	O	**PORT SELECT ENABLE:** This signal used in conjunction with PSEL provides contention-free port exchange on the data bus. When PSEN is low, port selection is allowed to change state.
WE	15	O	**WRITE ENABLE:** This signal provides the dynamic RAM array the write enable input for a write operation.

Table 1. Pin Description (Continued)

Symbol	Pin	Type	Name and Function
$\overline{FWR}$	16	I	**FULL WRITE:** This is an ECC input signal that instructs the 8207, in an ECC configuration, whether the present write cycle is normal RAM write (full write) or a RAM partial write (read-modify-write) cycle.
RESET	17	I	**RESET:** This signal causes all internal counters and state flip-flops to be reset and upon release of RESET, data appearing at the PDI pin is clocked in by the PCLK output. The states of the PDI, PCTLA, PCTLB and RFRQ pins are sampled by RESET going inactive and are used to program the 8207. An 8-cycle dynamic RAM warm-up is performed after clocking PDI bits into the 8207.
$\overline{CAS0}$ $\overline{CAS1}$ $\overline{CAS2}$ $\overline{CAS3}$	18 19 20 21	O O O O	**COLUMN ADDRESS STROBE:** These outputs are used by the dynamic RAM array to latch the column address, present on the AO0-8 pins. These outputs are selected by the BS0 and BS1 as programmed by program bits RB0 and RB1. These outputs drive the dynamic RAM array directly and need no external drivers.
$\overline{RAS0}$ $\overline{RAS1}$ $\overline{RAS2}$ $\overline{RAS3}$	22 23 24 25	O O O O	**ROW ADDRESS STROBE:** These outputs are used by the dynamic RAM array to latch the row address, present on the AO0-8 pins. These outputs are selected by the BS0 and BS1 as programmed by program bits RB0 and RB1. These outputs drive the dynamic RAM array directly and need no external drivers.
V_{SS}	26 60	I I	**DRIVER GROUND:** Provides a ground for the output drivers. **LOGIC GROUND:** Provides a ground for the remainder of the device.
AO0 AO1 AO2 AO3 AO4 AO5 AO6 AO7 AO8	35 34 33 32 31 30 29 28 27	O O O O O O O O O	**ADDRESS OUTPUTS:** These outputs are designed to provide the row and column addresses of the selected port to the dynamic RAM array. These outputs drive the dynamic RAM array directly and need no external drivers.
BS0 BS1	36 37	I I	**BANK SELECT:** These inputs are used to select one of four banks of the dynamic RAM array as defined by the program bits RB0 and RB1.
AL0 AL1 AL2 AL3 AL4 AL5 AL6 AL7 AL8	38 39 40 41 42 44 45 46 47	I I I I I I I I I	**ADDRESS LOW:** These lower-order address inputs are used to generate the row address for the internal address multiplexer.
AH0 AH1 AH2 AH3 AH4 AH5 AH6 AH7 AH8	48 49 50 51 52 53 54 55 56	I I I I I I I I I	**ADDRESS HIGH:** These higher-order address inputs are used to generate the column address for the internal address multiplexer.
PDI	57	I	**PROGRAM DATA INPUT:** This input programs the various user-selectable options in the 8207. The PCLK pin shifts programming data into the PDI input from optional external shift registers. This pin may be strapped high or low to a default ECC (PDI =Logic "I") or non-ECC (PDI = Logic "O") mode configuration.
RFRQ	58	I	**REFRESH REQUEST:** This input is sampled on the falling edge of RESET. If it is high at RESET, then the 8207 is programmed for internal refresh request or external refresh request with failsafe protection. If it is low at RESET, then the 8207 is programmed for external refresh without failsafe protection or burst refresh. Once programmed the RFRQ pin accepts signals to start an external refresh with failsafe protection or external refresh without failsafe protection or a burst refresh.

Table 1. Pin Description (Continued)

Symbol	Pin	Type	Name and Function
CLK	59	I	**CLOCK:** This input provides the basic timing for sequencing the internal logic.
$\overline{RDB}$	61	I	**READ FOR PORT B:** This pin is the read memory request command input for port B. This input also directly accepts the $\overline{S1}$ status line from Intel processors.
$\overline{WRB}$	62	I	**WRITE FOR PORT B:** This pin is the write memory request command input for port B. This input also directly accepts the $\overline{S0}$ status line from Intel processors.
$\overline{PEB}$	63	I	**PORT ENABLE FOR PORT B:** This pin serves to enable a RAM cycle request for port B. It is generally decoded from the port address.
PCTLB	64	I	**PORT CONTROL FOR PORT B:** This pin is sampled on the falling edge of RESET. It configures port B to accept command inputs or processor status inputs. If low after RESET, the 8207 is programmed to accept command or iAPX 286 status inputs or Multibus commands. If high after RESET, the 8207 is programmed to accept status inputs from iAPX 86 or iAPX 186 processors. The $\overline{S2}$ status line should be connected to this input if programmed to accept iAPX 86 or iAPX 186 status inputs. When programmed to accept commands or iAPX 286 status, it should be tied low or it may be used as a Multibus-compatible inhibit signal.
$\overline{RDA}$	65	I	**READ FOR PORT A:** This pin is the read memory request command input for port A. This input also directly accepts the $\overline{S1}$ status line from Intel processors.
$\overline{WRA}$	66	I	**WRITE FOR PORT A:** This pin is the write memory request command input for port A. This input also directly accepts the $\overline{S0}$ status line from Intel processors.
$\overline{PEA}$	67	I	**PORT ENABLE FOR PORT A:** This pin serves to enable a RAM cycle request for port A. It is generally decoded from the port address.
PCTLA	68	I	**PORT CONTROL FOR PORT A:** This pin is sampled on the falling edge of RESET. It configures port A to accept command inputs or processor status inputs. If low after RESET, the 8207 is programmed to accept command or iAPX 286 status inputs or Multibus commands. If high after RESET, the 8207 is programmed to accept status inputs from iAPX 86 or iAPX 186 processors. The $\overline{S2}$ status line should be connected to this input if programmed to accept iAPX 86 or iAPX 186 status inputs. When programmed to accept commands or iAPX 286 status, it should be tied low or it may be connected to INHIBIT when operating with Multibus.

GENERAL DESCRIPTION

The Intel 8207 Advanced Dynamic RAM Controller (ADRC) is a microcomputer peripheral device which provides the necessary signals to address, refresh and directly drive 16K, 64K and 256K dynamic RAMs. This controller also provides the necessary arbitration circuitry to support dual-port access of the dynamic RAM array.

The ADRC supports several microprocessor interface options including synchronous and asynchronous connection to iAPX 86, iAPX 88, iAPX 186, iAPX 188, iAPX 286 and Multibus.

This device may be used with the 8206 Error Detection and Correction Unit (EDCU). When used with the 8206, the 8207 is programmed in the Error Checking and Correction (ECC) mode. In this mode, the 8207 provides all the necessary control signals for the 8206 to perform memory initialization and transparent error scrubbing during refresh.

FUNCTIONAL DESCRIPTION

Processor Interface

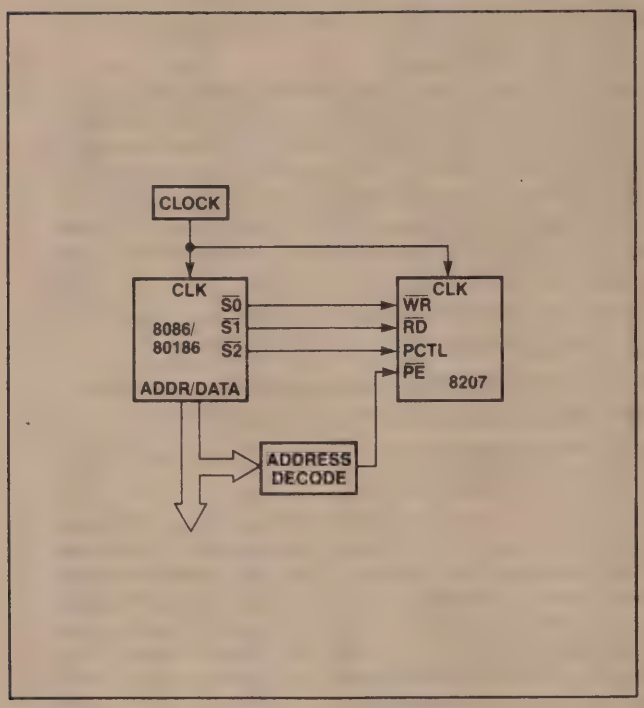
The 8207 has control circuitry for two ports each capable of supporting one of several possible bus structures. The ports are independently configurable allowing the dynamic RAM to serve as an interface between two different bus structures.

Each port of the 8207 may be programmed to run synchronous or asynchronous to the processor clock. (See Synchronous/Asynchronous Mode) The 8207 has been optimized to run synchronously with Intel's iAPX 86, iAPX 88, iAPX 186, iAPX 188 and iAPX 286. When the 8207 is programmed to run in asynchronous mode, the 8207 inserts the necessary synchronization circuitry for the $\overline{RD}$, $\overline{WR}$, $\overline{PE}$, and PCTL inputs.

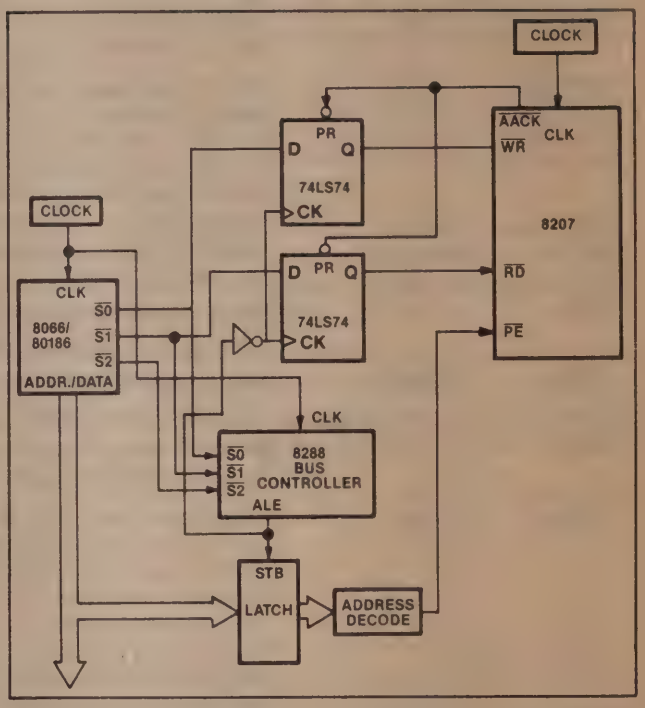
The 8207 achieves high performance (i.e. no wait states) by decoding the status lines directly from the iAPX 86, iAPX 88, iAPX 186, iAPX 188 and iAPX 286 processors. The 8207 can also be programmed to receive read or write Multibus commands or commands from a bus controller. (See Status/Command Mode)

The 8207 may be programmed to accept the clock of the iAPX 86, 88, 186, 188, or 286. The 8207 adjusts its internal timing to allow for the different clock frequencies of these microprocessors. (See Microprocessor Clock Frequency Option)

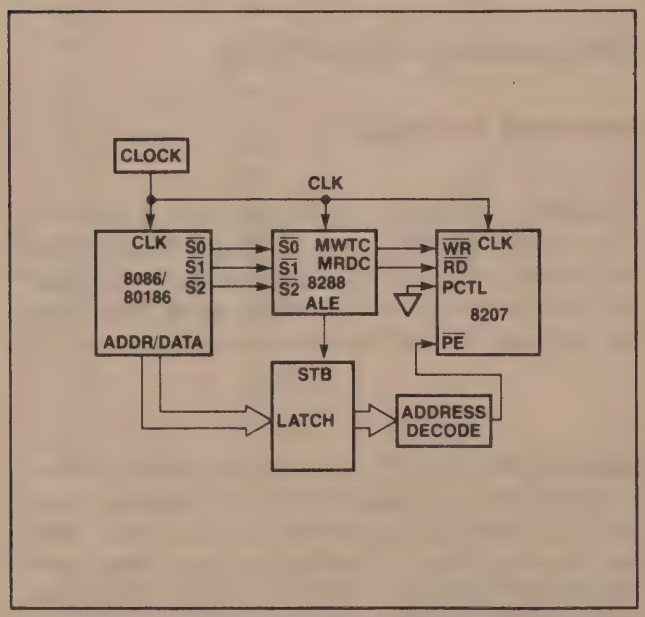
Figure 2 shows the different processor interfaces to the 8207 using the synchronous or asynchronous mode and status or command interface.



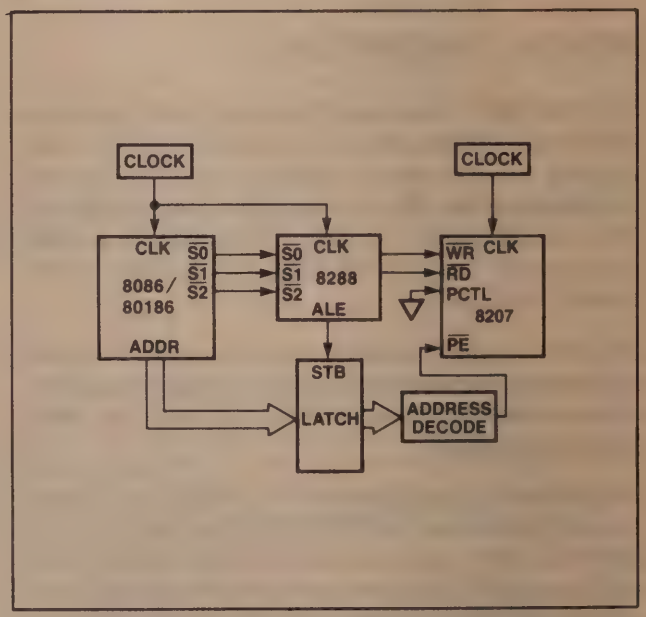
Slow-Cycle Synchronous-Status Interface



Slow-Cycle Asynchronous-Status Interface

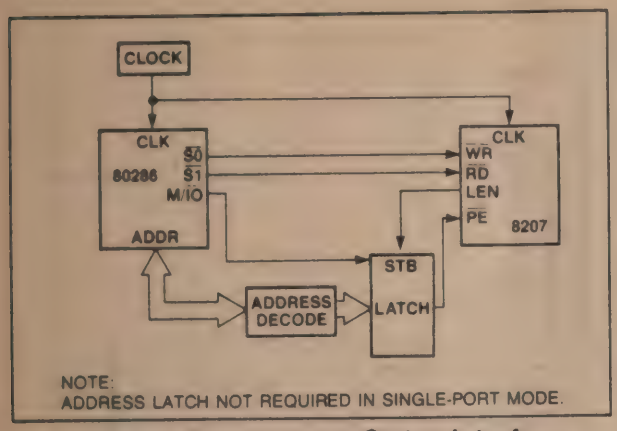


Slow-Cycle Synchronous-Command Interface

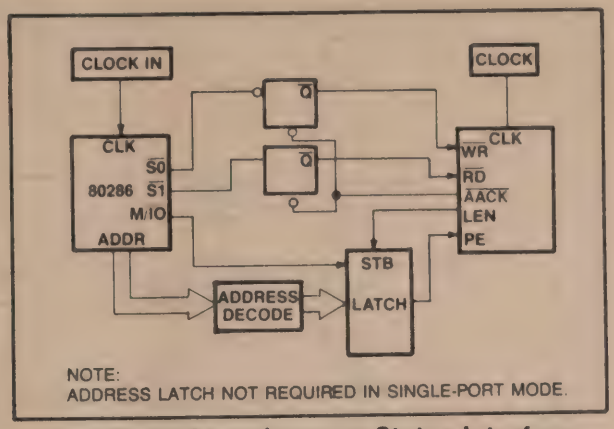


Slow-Cycle Asynchronous-Command Interface

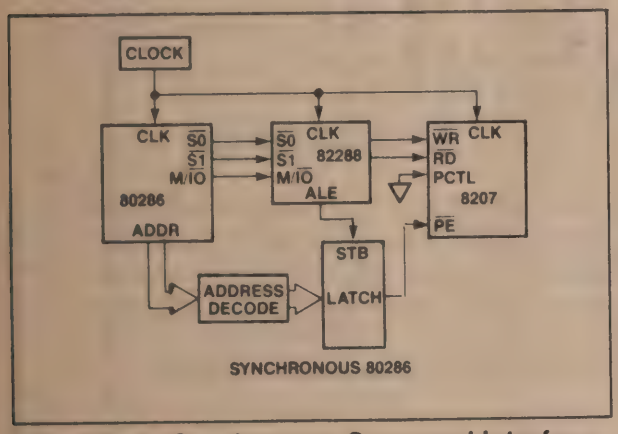
Figure 2A. Slow-cycle (CFS=0) Port Interfaces Supported by the 8207



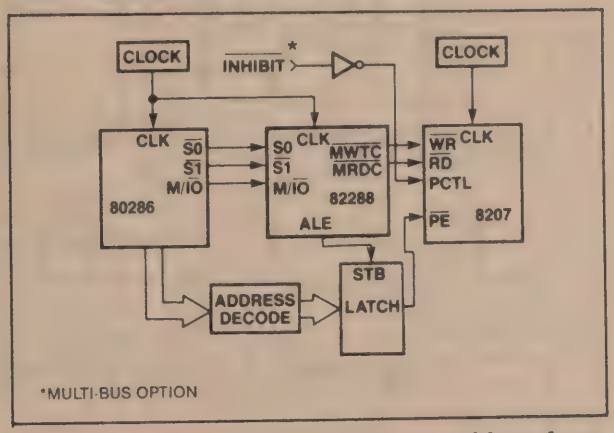
Fast-Cycle Synchronous-Status Interface



Fast-Cycle Asynchronous-Status Interface



Fast-Cycle Synchronous-Command Interface



Fast-Cycle Asynchronous-Command Interface

Figure 2B. Fast-cycle (CFS=1) Port Interfaces Supported by the 8207

Single-Port Operation

The use of an address latch with the iAPX 286 status interface is not needed since the 8207 can internally latch the addresses with an internal signal similar in behavior to the LEN output. This operation is active only in single-port applications when the processor is interfaced to port A.

Dual-Port Operation

The 8207 provides for two-port operation. Two independent processors may access memory controlled by the 8207. The 8207 arbitrates between each of the processor requests and directs data to or from the appropriate port. Selection is done on a priority concept that reassigns priorities based upon past history. Processor requests are internally queued.

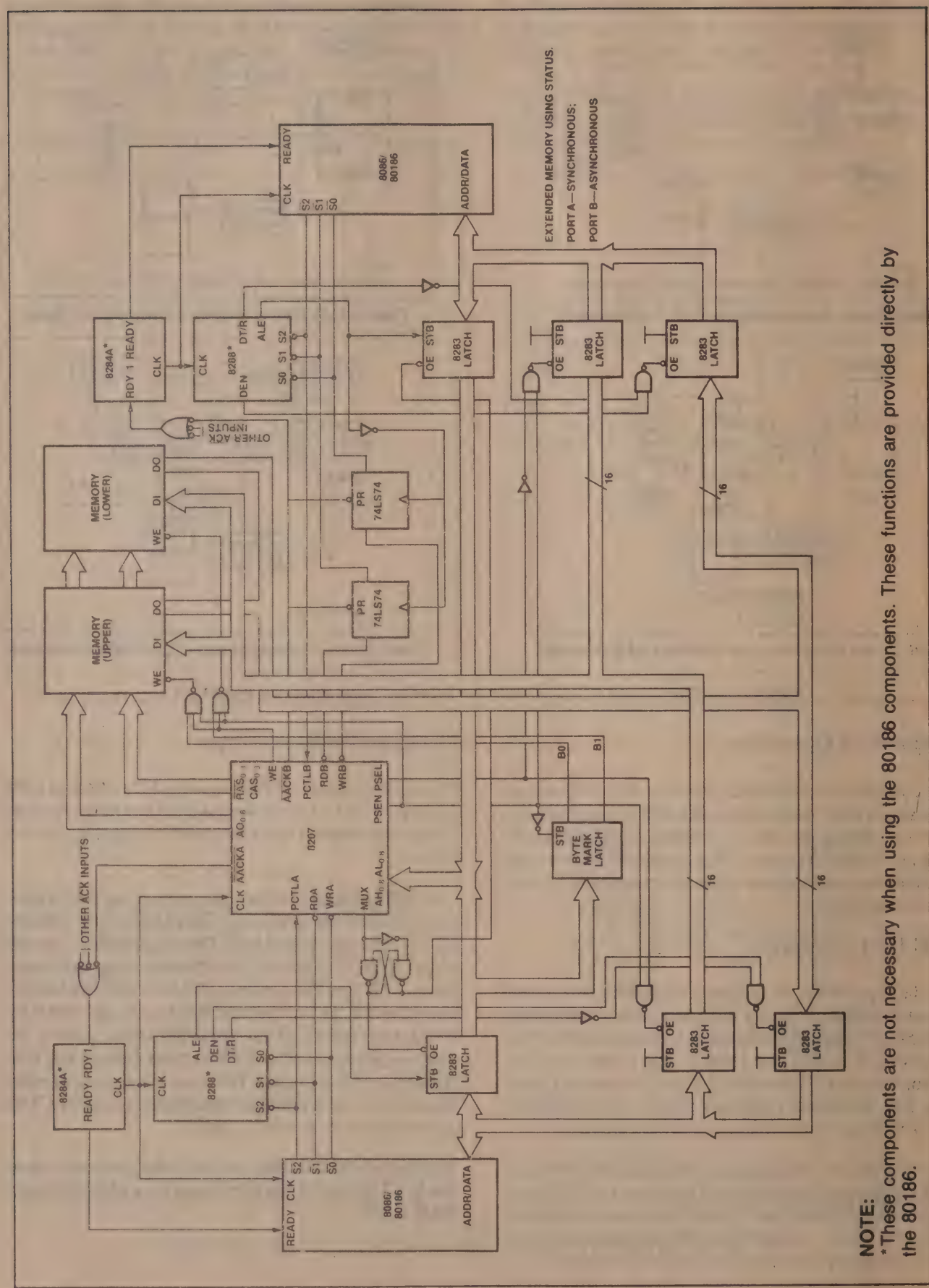
Figure 3 shows a dual-port configuration with two iAPX 86 systems interfacing to dynamic RAM. One of the processor systems is interfaced synchronously using the status interface and the other is interfaced asynchronously also using the status interface.

Dynamic RAM Interface

The 8207 is capable of addressing 16K, 64K and 256K dynamic RAMs. Figure 4 shows the connection of the processor address bus to the 8207 using the different RAMs.

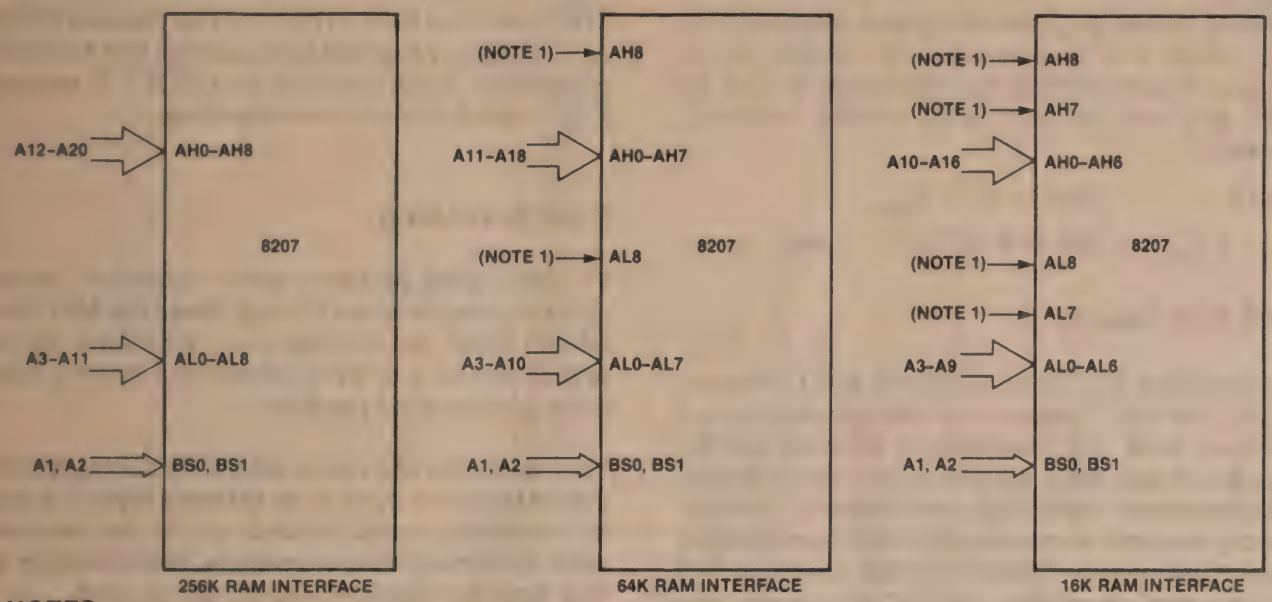
The 8207 divides memory into as many as four banks, each bank having its own Row ($\overline{RAS}$) and Column ($\overline{CAS}$) Address Strobe pair. This organization permits RAM cycle interleaving and permits error scrubbing during ECC refresh cycles. RAM cycle interleaving overlaps the start of the next RAM cycle with the RAM Precharge period of the previous cycle. Hiding the precharge period of one RAM cycle behind the data access period of the next RAM cycle optimizes memory bandwidth and is effective as long as successive RAM cycles occur in alternate banks.

Successive data access to the same bank will cause the 8207 to wait for the precharge time of the previous RAM cycle.



NOTE:
*These components are not necessary when using the 80186 components. These functions are provided directly by the 80186.

Figure 3. 8086/80186 Dual Port System



NOTES:

(1) Unassigned address input pins should be strapped high or low.

(2) A0 along with BHE are used to select a byte within a processor word.

(3) Low order address bits are used as bank select inputs so that consecutive memory access requests are to alternate banks allowing bank interleaving of memory cycles.

Figure 4. Processor Address Interface to the 8207 Using 16K, 64K, and 256K RAMS

If not all RAM banks are occupied, the 8207 reassigns the $\overline{RAS}$ and $\overline{CAS}$ strobes to allow using wider data words without increasing the loading on the $\overline{RAS}$ and $\overline{CAS}$ drivers. Table 2 shows the bank selection decoding and the word expansion, including $\overline{RAS}$ and $\overline{CAS}$ assignments. For example, if only two RAM banks are occupied, then two $\overline{RAS}$ and two $\overline{CAS}$ strobes are activated per bank. Program bits RB1 and RB0 are not used to check the bank select inputs BS1 and BS0. The system design must protect from accesses to "illegal", non-existent banks of memory, by deactivating the PEA, PEB inputs when addressing an illegal bank.

The 8207 can interface to fast or slow RAMs. The 8207 adjusts and optimizes internal timings for either the fast or slow RAMs as programmed. (See RAM Speed Option.)

Memory Initialization

After programming, the 8207 performs eight RAM "warm-up" cycles to prepare the dynamic RAM for proper device operation. During "warm-up" some RAM parameters, such as tRAH, tASC, may not be met. This causes no harm to the dynamic RAM array. If configured for operation with error correction, the 8207 and 8206 EDCU will proceed to initialize all of memory (memory is written with zeros with corresponding check bits).

Table 2. Bank Selection Decoding and Word Expansion

Program Bits		Bank Input		
RB1	RB0	BS1	BSØ	RAS/CAS Pair Allocation
0	0	0	0	RAS_{0-3}, CAS_{0-3} to Bank 0
0	0	0	1	Illegal
0	0	1	0	Illegal
0	0	1	1	Illegal
0	1	0	0	$RAS_{0,1}$, $CAS_{0,1}$ to Bank 0
0	1	0	1	$RAS_{2,3}$, $CAS_{2,3}$ to Bank 1
0	1	1	0	Illegal
0	1	1	1	Illegal
1	0	0	0	RAS_0, CAS_0 to Bank 0
1	0	0	1	RAS_1, CAS_1 to Bank 1
1	0	1	0	RAS_2, CAS_2 to Bank 2
1	0	1	1	Illegal
1	1	0	0	RAS_0, CAS_0 to Bank 0
1	1	0	1	RAS_1, CAS_1 to Bank 1
1	1	1	0	RAS_2, CAS_2 to Bank 2
1	1	1	1	RAS_3, CAS_3 to Bank 3

Because the time to initialize memory is fairly long, the 8207 may be programmed to skip initialization in ECC mode. The time required to initialize all of memory is dependent on the clock cycle time to the 8207 and can be calculated by the following equation:

eq.1 $T_{INIT} = (2^{23})\ T_{CLCL}$

if T_{CLCL} = 125 ns then $T_{INIT} \approx 1$ sec.

8206 ECC Interface

For operation with Error Checking and Correction (ECC), the 8207 adjusts its internal timing and changes some pin functions to optimize performance and provide a clean dual-port memory interface between the 8206 EDCU and memory. The 8207 directly supports a master-only (16-bit word plus 6 check bits) system. Under extended operation and reduced clock frequency, the 8207 will support any ECC master-slave configuration up to 80 data bits, which is the maximum set by the 8206 EDCU. (See Extend Option)

Correctable errors detected during memory read cycles are corrected immediately and then written back into memory.

In a synchronous bus environment, ECC system performance has been optimized to enhance processor throughput, while in an asynchronous bus environment (the Multibus), ECC performance has been optimized to get valid data onto the bus as quickly as possible. Performance optimization, processor throughput or quick data access may be selected via the Transfer Acknowledge Option.

The main difference between the two ECC implementations is that, when optimized for processor throughput, RAM data is always corrected and an advanced transfer acknowledge is issued at a point when, by knowing the processor characteristics, data is guaranteed to be valid by the time the processor needs it.

When optimized for quick data access, (valid for Multibus) the 8206 is configured in the uncorrecting mode where the delay associated with error correction circuitry is transparent, and a transfer acknowledge is issued as soon as valid data is known to exist. If the $\overline{ERROR}$ flag is activated, then the transfer acknowledge is delayed until after the 8207 has instructed the 8206 to correct the data and the corrected data becomes available on the bus. Figure 5 illustrates a dual-port ECC system.

Figure 6 illustrates the interface required to drive the $\overline{CRCT}$ pin of the 8206, in the case that one port (PORT A) receives an advanced acknowledge (not Multibus-compatible), while the other port (PORT B) receives $\overline{XACK}$ (which is Multibus-compatible).

Error Scrubbing

The 8207/8206 performs error correction during refresh cycles (error scrubbing). Since the 8207 must refresh RAM, performing error scrubbing during refresh allows it to be accomplished without additional performance penalties.

Upon detection of a correctable error during refresh, the RAM refresh cycle is lengthened slightly to permit the 8206 to correct the error and for the corrected word to be rewritten into memory. Uncorrectable errors detected during scrubbing are ignored.

Refresh

The 8207 provides an internal refresh interval counter and a refresh address counter to allow the 8207 to refresh memory. The 8207 will refresh 128 rows every 2 milliseconds or 256 rows every 4 milliseconds, which allows all RAM refresh options to be supported. In addition, there exists the ability to refresh 256 row address locations every 2 milliseconds via the Refresh Period programming option.

The 8207 may be programmed for any of four different refresh options: Internal refresh only, External refresh with failsafe protection, External refresh without failsafe protection, Burst Refresh mode, or no refresh. (See Refresh Options)

It is possible to decrease the refresh time interval by 10%, 20% or 30%. This option allows the 8207 to compensate for reduced clock frequencies. Note that an additional 5% interval shortening is built-in in all refresh interval options to compensate for clock variations and non-immediate response to the internally generated refresh request. (See Refresh Period Options)

External Refresh Requests after RESET

External refresh requests are not recognized by the 8207 until after it is finished programming and preparing memory for access. Memory preparation includes 8 RAM cycles to prepare and ensure proper

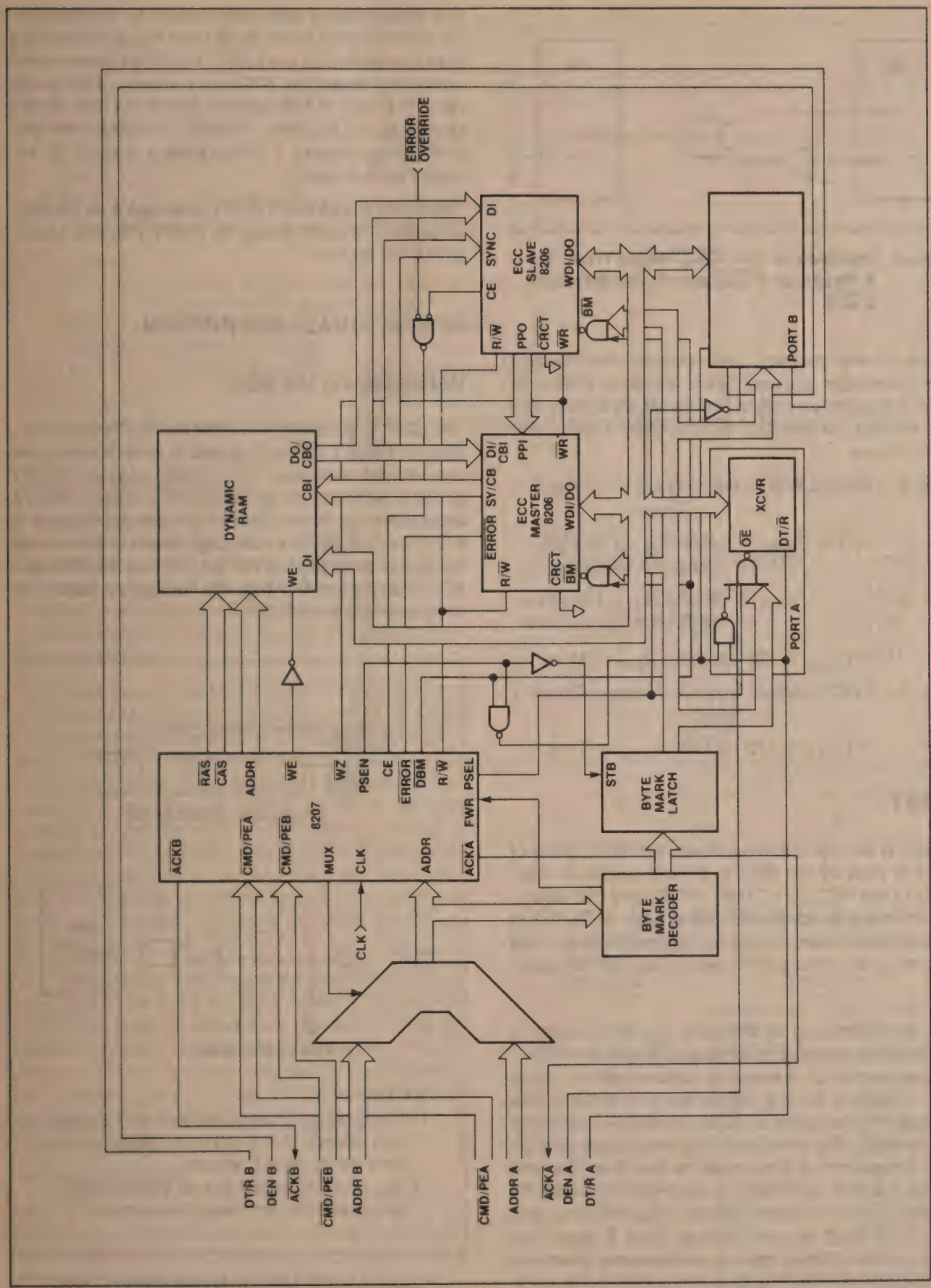


Figure 5. Two-Port ECC Implementation Using the 8207 and the 8206

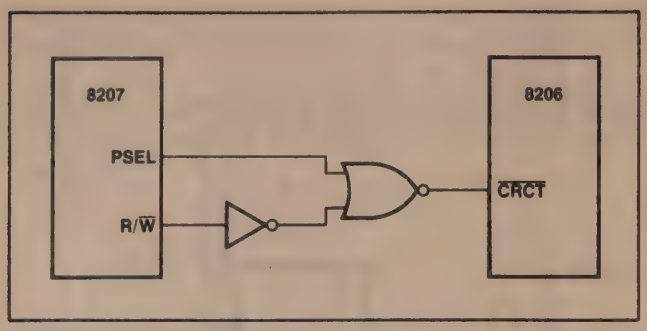


Figure 6. Interface to 8206 $\overline{CRCT}$ Input When Port A Receives $\overline{AACK}$ and Port B Receives $\overline{XACK}$

dynamic RAM operation, and memory initialization if error correction is used. Many dynamic RAMs require this warm-up period for proper operation. The time it takes for the 8207 to recognize a request is shown below.

eq. 2 Non-ECC Systems: $T_{RESP} = T_{PROG} + T_{PREP}$

eq. 3 where: $T_{PROG} = (66)(T_{CLCL})$ which is programming time

eq. 4 $T_{PREP} = (8)(32)(T_{CLCL})$ which is the RAM warm-up time

if $T_{CLCL} = 125$ ns then $T_{RESP} \approx 41$ us

eq. 5 ECC Systems: $T_{RESP} = T_{PROG} + T_{PREP} + T_{INIT}$

if $T_{CLCL} = 125$ ns then $T_{RESP} \approx 1$ sec

RESET

RESET is an asynchronous input, the falling edge of which is used by the 8207 to directly sample to logic levels of the PCTLA, PCTLB, RFRQ, and PDI inputs. The internally synchronized falling edge of RESET is used to begin programming operations (shifting in the contents of the external shift register into the PDI input).

Until programming is complete the 8207 registers but does not respond to command or status inputs. A simple means of preventing commands or status from occurring during this period is to differentiate the system reset pulse to obtain a smaller reset pulse for the 8207. The total time of the reset pulse and the 8207 programming time must be less than the time before the first command in systems that alter the default port synchronization programming bits (default is Port A synchronous, Port B asynchronous). Differentiated reset is unnecessary when the default port synchronization programming is used.

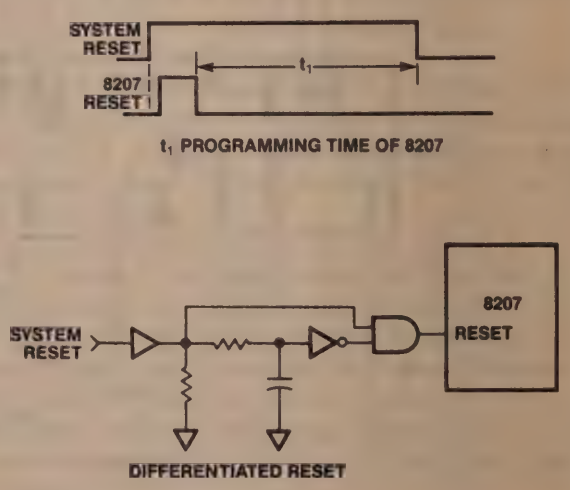
The differentiated reset pulse would be shorter than the system reset pulse by at least the programming period required by the 8207. The differentiated reset pulse first resets the 8207, and system reset would reset the rest of the system. While the rest of the system is still in reset, the 8207 completes its programming. Figure 7 illustrates a circuit to accomplish this task.

Within four clocks after RESET goes active, all the 8207 outputs will go high, except for PSEN, WE, and AO0-2, which will go low.

OPERATIONAL DESCRIPTION

Programming the 8207

The 8207 is programmed after reset. On the falling edge of RESET, the logic states of several input pins are latched internally. The falling edge of RESET actually performs the latching, which means that the logic levels on these inputs must be stable prior to that time. The inputs whose logic levels are latched at the end of reset are the PCTLA, PCTLB, REFRQ, and PDI pins. Figure 8 shows the necessary timing for programming the 8207.

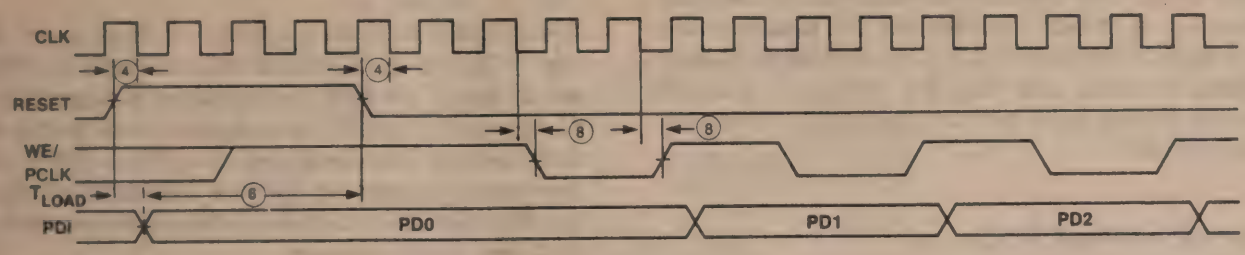


NOTES:

(1) Required only when the port synchronization options (SA & SB) are altered from their initial default values.

(2) V_{CC} must be stable before system reset is activated when using this circuit.

Figure 7. 8207 Differentiated Reset Circuit



NOTES:
TRTVCL — Reset is an asynchronous input, if reset occurs before T_1, then it is guaranteed to be recognized.
TPGVCL — Minimum PDI valid time prior to reset going low.
TCLPC — MUX/PCLK delay.
TLOAD — Asynchronous load data propagation delay.

Figure 8. Timing Illustrating External Shift Register Requirements for Programming the 8207

Status/Command Mode

The two processor ports of the 8207 are configured by the states of the PCTLA and PCTLB pins. Which interface is selected depends on the state of the individual port's PCTL pin at the end of reset. If PCTL is high at the end of the reset, the 8086 Status interface is selected; if it is low, then the Command interface is selected.

The status lines of the 80286 are similar in code and timing to the Multibus command lines, while the status code and timing of the 8086 and 8088 are identical to those of the 80186 and 80188 (ignoring the differences in clock duty cycle). Thus there exists two interface configurations, one for the 80286 status or Multibus memory commands, which is called the Command interface, and one for 8086, 8088, 80186 or 80188 status, called the 8086 Status interface. The Command interface can also directly interface to the command lines of the bus controllers for the 8086, 8088, 80186 and the 80286.

The 8086 Status interface allows direct decoding of the status of the iAPX 86, iAPX 88, iAPX 186 and the iAPX 188. Table 3 shows how the status lines are decoded. While in the Command mode the iAPX 286 status can be directly decoded. Microprocessor bus controller read or write commands or Multibus commands can also be directed to the 8207 when in Command mode.

Refresh Options

Immediately after system reset, the state of the REFRQ input pin is examined. If REFRQ is high, the 8207 provides the user with the choice between self-refresh or user-generated refresh with failsafe protection. Failsafe protection guarantees that if the

Table 3A. Status Coding of 8086, 80186 and 80286

Status Code			Function	
$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	8086/80186	80286
0	0	0	INTERRUPT	INTERRUPT
0	0	1	I/O READ	I/O READ
0	1	0	I/O WRITE	I/O WRITE
0	1	1	HALT	IDLE
1	0	0	INSTRUCTION FETCH	HALT
1	0	1	MEMORY READ	MEMORY READ
1	1	0	MEMORY WRITE	MEMORY WRITE
1	1	1	IDLE	IDLE

Table 3B. 8207 Response

8207 Command			Function	
PCTL	$\overline{RD}$	$\overline{WR}$	8086/80186 Status Interface	80286 Status or Command Interface
0	0	0	IGNORE	IGNORE*
0	0	1	IGNORE	READ
0	1	0	IGNORE	WRITE
0	1	1	IGNORE	IGNORE*
1	0	0	READ	IGNORE
1	0	1	READ	INHIBIT
1	1	0	WRITE	INHIBIT
1	1	1	IGNORE	IGNORE

*Illegal with CFS=0

user does not come back with another refresh request before the internal refresh interval counter times out, a refresh request will be automatically generated. If the REFRQ pin is low immediately after a reset, then the user has the choice of a single external refresh cycle without failsafe, burst refresh or no refresh.

Internal Refresh Only

For the 8207 to generate internal refresh requests, it is necessary only to strap the REFRQ input pin high.

External Refresh with Failsafe

To allow user-generated refresh requests with failsafe protection, it is necessary to hold the REFRQ input high until after reset. Thereafter, a low-to-high transition on this input causes a refresh request to be generated and the internal refresh interval counter to be reset. A high-to-low transition has no effect on the 8207. A refresh request is not recognized until a previous request has been serviced.

External Refresh without Failsafe

To generate single external refresh requests without failsafe protection, it is necessary to hold REFRQ low until after reset. Thereafter, bringing REFRQ high for one clock period causes a refresh request to be generated. A refresh request is not recognized until a previous request has been serviced.

Burst Refresh

Burst refresh is implemented through the same procedure as a single external refresh without failsafe (i.e., REFRQ is kept low until after reset). Thereafter, bringing REFRQ high for at least two clock periods causes a burst of up to 128 row address locations to be refreshed.

In ECC-configured systems, 128 locations are scrubbed. Any refresh request is not recognized until a previous request has been serviced (i.e., burst completed).

No Refresh

It is necessary to hold REFRQ low until after reset. This is the same as programming External Refresh without Failsafe. No refresh is accomplished by keeping REFRQ low.

Option Program Data Word

The program data word consists of 16 program data bits, PD0—PD15. If the first program data bit PD0 is set to logic 1, the 8207 is configured to support ECC. If it is logic 0, the 8207 is configured to support a non-ECC system. The remaining bits, PD1—PD15, may then be programmed to optimize a selected configuration. Figures 9 and 10 show the Program words for non-ECC and ECC operation.

Using an External Shift Register

The 8207 may be configured to use an external shift register with asynchronous load capability such as a 74LS165. The reset pulse serves to parallel load the shift register and the 8207 supplies the clocking signal to shift the data in. Figure 11 shows a sample circuit diagram of an external shift register circuit.

Serial data is shifted into the 8207 via the PDI pin (57), and clock is provided by the MUX/PCLK pin (12), which generates a total of 16 clock pulses. After programming is complete, data appearing at the input of the PDI pin is ignored. MUX/PCLK is a dual-function pin. During programming, it serves to clock the external shift register, and after programming is completed, it reverts to a MUX control pin. As the pin changes state to select different port addresses, it continues to clock the shift register. This does not present a problem because data at the PDI pin is ignored after programming. Figure 8 illustrates the timing requirements of the shift register circuitry.

ECC Mode (ECC Program Bit)

The state of PDI (Program Data In) pin at reset determines whether the system is an ECC or non-ECC configuration. It is used internally by the 8207 to begin configuring timing circuits, even before programming is completely finished. The 8207 then begins programming the rest of the options.

Default Programming Options

After reset, the 8207 serially shifts in a program data word via the PDI pin. This pin may be strapped either high or low, or connected to an external shift register. Strapping PDI high causes the 8207 to default to a particular system configuration with error correction, and strapping it low causes the 8207 to default to a particular system configuration without error correction. Table 4 shows the default configurations.

PD15							PD8	PD7						PD0
0	0	TM 1	$\overline{PPR}$	$\overline{FFS}$	EXT	$\overline{PLS}$	CI0	CI1	$\overline{RB1}$	$\overline{RB0}$	$\overline{RFS}$	$\overline{CFS}$	SB	$\overline{SA}$	0

PROGRAM DATA BIT	NAME	POLARITY/FUNCTION
PD0	ECC	ECC=0 FOR NON-ECC MODE
PD1	$\overline{SA}$	$\overline{SA}$=0 PORT A IS SYNCHRONOUS $\overline{SA}$=1 PORT A IS ASYNCHRONOUS
PD2	SB	SB=0 PORT B IS ASYNCHRONOUS SB=1 PORT B IS SYNCHRONOUS
PD3	$\overline{CFS}$	$\overline{CFS}$=0 FAST-CYCLE iAPX 286 MODE $\overline{CFS}$=1 SLOW-CYCLE iAPX 86 MODE
PD4	$\overline{RFS}$	$\overline{RFS}$=0 FAST RAM $\overline{RFS}$=1 SLOW RAM
PD5 PD6	$\overline{RB0}$ $\overline{RB1}$	RAM BANK OCCUPANCY SEE TABLE 2
PD7 PD8	CI1 CI0	COUNT INTERVAL BIT 1; SEE TABLE 6 COUNT INTERVAL BIT 0; SEE TABLE 6
PD9	$\overline{PLS}$	$\overline{PLS}$=0 LONG REFRESH PERIOD $\overline{PLS}$=1 SHORT REFRESH PERIOD
PD10	EXT	EXT=0 NOT EXTENDED EXT=1 EXTENDED
PD11	$\overline{FFS}$	$\overline{FFS}$=0 FAST CPU FREQUENCY $\overline{FFS}$=1 SLOW CPU FREQUENCY
PD12	$\overline{PPR}$	$\overline{PPR}$=0 MOST RECENTLY USED PORT PRIORITY $\overline{PPR}$=1 PORT A PREFERRED PRIORITY
PD13	TM1	TM1=0 TEST MODE 1 OFF TM1=1 TEST MODE 1 ENABLED
PD14	0	RESERVED MUST BE ZERO
PD15	0	RESERVED MUST BE ZERO

Figure 9. Non-ECC Mode Program Data Word

PD15							PD8	PD7						PD0
$\overline{TM2}$	RB1	RB0	PPR	FFS	$\overline{EXT}$	PLS	$\overline{CI0}$	$\overline{CI1}$	XB	$\overline{XA}$	RFS	CFS	$\overline{SB}$	SA	1

PROGRAM DATA BIT	NAME	POLARITY/FUNCTION
PD0	ECC	ECC=1 ECC MODE
PD1	SA	SA=0 PORT A ASYNCHRONOUS SA=1 PORT A SYNCHRONOUS
PD2	$\overline{SB}$	$\overline{SB}$=0 PORT B SYNCHRONOUS $\overline{SB}$=1 PORT B ASYNCHRONOUS
PD3	CFS	CFS=0 SLOW-CYCLE iAPX 86 MODE CFS=1 FAST-CYCLE iAPX 286 MODE
PD4	RFS	RFS=0 SLOW RAM RFS=1 FAST RAM
PD5	$\overline{XA}$	$\overline{XA}$=0 MULTIBUS-COMPATIBLE ACKA $\overline{XA}$=1 ADVANCED ACKA NOT MULTIBUS-COMPATIBLE
PD6	XB	XB=0 ADVANCED ACKB NOT MULTIBUS COMPATIBLE XB=1 MULTIBUS-COMPATIBLE ACKB
PD7 PD8	$\overline{CI1}$ $\overline{CI0}$	COUNT INTERVAL BIT 1; SEE TABLE 6 COUNT INTERVAL BIT 0; SEE TABLE 6
PD9	PLS	PLS=0 SHORT REFRESH PERIOD PLS=1 LONG REFRESH PERIOD
PD10	$\overline{EXT}$	$\overline{EXT}$=0 MASTER AND SLAVE EDCU $\overline{EXT}$=1 MASTER EDCU ONLY
PD11	FFS	FFS=0 SLOW CPU FREQUENCY FFS=1 FAST CPU FREQUENCY
PD12	PPR	PPR=0 PORT A PREFERRED PRIORITY PPR=1 MOST RECENTLY USED PORT PRIORITY
PD13 PD14	RB0 RB1	RAM BANK OCCUPANCY SEE TABLE 2
PD15	$\overline{TM2}$	$\overline{TM2}$=0 TEST MODE 2 ENABLED $\overline{TM2}$=1 TEST MODE 2 OFF

Figure 10. ECC Mode Program Data Word

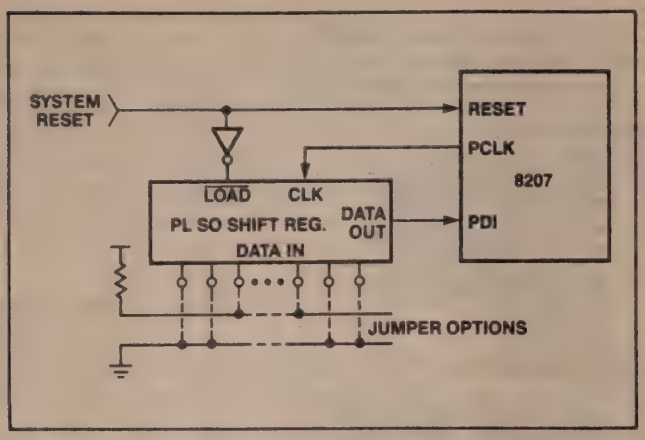


Figure 11. External Shift Register Interface

Table 4A.
Default Non-ECC Programming, PDI Pin (57) Tied to Ground.

Port A is Synchronous ($\overline{EAACKA}$ and $\overline{XACKA}$)
Port B is Asynchronous ($\overline{LAACKB}$ and $\overline{XACKB}$)
Fast-cycle Processor Interface (iAPX 286)
Fast RAM
Refresh Interval uses 236 clocks
128 Row refresh in 2 ms; 256 Row refresh in 4 ms
Fast Processor Clock Frequency (16 MHz)
"Most Recently Used" Priority Scheme
4 RAM banks occupied

Table 4B.
Default ECC Programming, PDI Pin (57) Tied to V_{CC}.

Port A is Synchronous
Port B is Asynchronous
Fast-cycle Processor Interface (iAPX 286)
Fast RAM
Port A has $\overline{EAACKA}$ strobe (non-multibus)
Port B has $\overline{XACKB}$ strobe (multibus)
Refresh interval uses 236 clocks
128 Row refresh in 2 ms; 256 Row refresh in 4 ms
Master EDCU only (16-bit system)
Fast Processor Clock Frequency (16 MHz)
"Most Recently Used" Priority Scheme
4 RAM banks occupied

If further system flexibility is needed, one or two external shift registers can be used to tailor the 8207 to its operating environment.

Synchronous/Asynchronous Mode (SA and SB Program Bits)

Each port of the 8207 may be independently configured to accept synchronous or asynchronous port commands ($\overline{RD}$, $\overline{WR}$, PCTL) and Port Enable ($\overline{PE}$) via the program bits SA and SB. The state of the SA and SB programming bits determine whether their associated ports are synchronous or asynchronous.

While a port may be configured with either the Status or Command interface in the synchronous mode, certain restrictions exist in the asynchronous mode. An asynchronous Command interface using the control lines of the Multibus is supported, and an asynchronous 8086 interface using the control lines of the 8086 is supported, with the use of TTL gates as illustrated in Figure 2. In the 8086 case, the TTL gates are needed to guarantee that status does not appear at the 8207's inputs too much before address, so that a cycle would start before address was valid.

Microprocessor Clock Frequency Option (CFS and FFS Program Bits)

The 8207 can be programmed to interface with slow-cycle microprocessors like the 8086, 8088, 80188 and 80186 or fast-cycle microprocessors like the 80286. The CFS bit configures the microprocessor interface to accept slow or fast cycle signals from either microprocessor group.

The FFS bit is used to select the speed of the microprocessor clock. Table 5 shows the various microprocessor clock frequency options that can be programmed.

Table 5.
Microprocessor Clock Frequency Options

Program Bits		Processor	Clock Frequency
CFS	FFS		
0	0	iAPX 86, 88, 186, 188	6 MHz
0	1	iAPX 86, 88, 186, 188	8 MHz
1	0	iAPX 286	12MHz
1	1	iAPX 286	16 MHz

The external clock frequency must be programmed so that the failsafe refresh repetition circuitry can adjust its internal timing accordingly to produce a refresh request as programmed.

RAM Speed Option (RFS Program Bit)

The RAM Speed programming option determines whether RAM timing will be optimized for a fast or slow RAM.

Refresh Period Options (CI0, CI1, and PLS Program Bits)

The 8207 refreshes with either 128 rows every 2 milliseconds or 256 rows every 4 milliseconds. This translates to one refresh cycle being executed approximately once every 15.6 microseconds. This rate can be changed to 256 rows every 2 milliseconds or a refresh approximately once every 7.8 microseconds via the Period Long/Short, program bit PLS, programming option. The 7.8 microsecond refresh request rate is intended for those RAMs, 64K and above, which may require a faster refresh rate.

In addition to PLS program option, two other programming bits for refresh exist: Count Interval 0 (CI0) and Count Interval 1 (CI1). These two programming bits allow the rate at which refresh requests are generated to be increased in order to permit refresh requests to be generated close to the same 15.6 or 7.8 microsecond period when the 8207 is operating at reduced frequencies. The interval between refreshes is decreased by 0%, 10%, 20%, or 30% as a function of how the count interval bits are programmed. A 5% guardband is built-in to allow for any clock frequency variations. Table 6 shows the refresh period options available.

The numbers tabulated under Count Interval represent the number of clock periods between internal refresh requests. The percentages in parentheses represent the decrease in the interval between refresh requests. Note that all intervals have a built-in 5% (approximately) safety factor to compensate for minor clock frequency deviations and non-immediate response to internal refresh requests.

Extend Option (EXT Program Bit)

The Extend option lengthens the memory cycle to allow longer access time which may be required by the system. Extend alters the RAM timing to compensate for increased loading on the Row and Column Address Strobes, and in the multiplexed Address Out lines.

Port Priority Option and Arbitration (PPR Program Bit)

The 8207 has to internally arbitrate among three ports: Port A, Port B and Port C—the refresh port. Port C is an internal port dedicated to servicing refresh requests, whether they are generated internally by the refresh inverval counter, or externally by the user. Two arbitration approaches are available via

Table 6. Refresh Count Interval Table

Ref. Period (μS)	CFS	PLS	FFS	Count Interval CI1, CI0 (8207 Clock Periods)			
				00 (0%)	01 (10%)	10 (20%)	11 (30%)
15.6	1	1	1	236	212	188	164
7.8	1	0	1	118	106	94	82
15.6	1	1	0	148	132	116	100
7.8	1	0	0	74	66	58	50
15.6	0	1	1	118	106	94	82
7.8	0	0	1	59	53	47	41
15.6	0	1	0	74	66	58	50
7.8	0	0	0	37	33	29	25

the Port Priority programming option, program bit PPR. PPR determines whether the most recently used port will remain selected (PPR = 1) or whether Port A will be favored or preferred over Port B (PPR = 0).

A port is selected if the arbiter has given the selected port direct access to the timing generators. The front-end logic, which includes the arbiter, is designed to operate in parallel with the selected port. Thus a request on the selected port is serviced immediately. In contrast, an unselected port only has access to the timing generators through the front-end logic. Before a RAM cycle can start for an unselected port, that port must first become selected (i.e., the MUX output now gates that port's address into the 8207 in the case of Port A or B). Also, in order to allow its address to stabilize, a newly selected port's first RAM cycle is started by the front-end logic. Therefore, the selected port has direct access to the timing generators. What all this means is that a request on a selected port is started immediately, while a request on an unselected port is started two to three clock periods after the request, assuming that the other two ports are idle. Under normal operating conditions, this arbitration time is hidden behind the RAM cycle of the selected port so that as soon as the present cycle is over a new cycle is started. Table 7 lists the arbitration rules for both options.

Port LOCK Function

The LOCK function provides each port with the ability to obtain uninterrupted access to a critical region of memory and, thereby, to guarantee that the opposite port cannot "sneak in" and read from or write to the critical region prematurely.

Only one LOCK pin is present and is multiplexed between the two ports as follows: when MUX is high, the 8207 treats the LOCK input as originating at PORT A, while when MUX is low, the 8207 treats LOCK as originating at PORT B. When the 8207 recognizes a LOCK, the MUX output will remain pointed to the locking port until LOCK is deactivated. Refresh is not affected by LOCK and can occur during a locked memory cycle.

Table 7. The Arbitration Rules for the Most Recently Used Port Priority and for Port A Priority Options Are As follows:

1.	If only one port requests service, then that port—if not already selected—becomes selected.
2a.	When no service requests are pending, the last selected processor port (Port A or B) will remain selected. (Most Recently Used Port Priority Option)
2b.	When no service requests are pending, Port A is selected whether it requests service or not. (Port A Priority Option)
3.	During reset initialization only Port C, the refresh port, is selected.
4.	If no processor requests are pending after reset initialization, Port A will be selected.
5a.	If Ports A and B simultaneously(*) request service while Port C is being serviced, then the next port to be selected is the one which was not selected prior to servicing Port C. (Most Recently Used Port Priority Option)
5b.	If Ports A and B simultaneously(*) request service while Port C is selected, then the next port to be selected is Port A. (Port A Priority Option)
6.	If a port simultaneously requests service with the currently selected port, service is granted to the selected port.
7.	The MUX output remains in its last state whenever Port C is selected.
8.	If Port C and either Port A or Port B (or both) simultaneously request service, then service is granted to the requester whose port is already selected. If the selected port is not requesting service, then service is granted to Port C.
9.	If during the servicing of one port, the other port requests service before or simultaneously with the refresh port, the refresh port is selected. A new port is not selected before the presently selected port is deactivated.
10.	Activating LOCK will mask off service requests from Port B if the MUX output is high, or from Port A if the MUX output is low.

* By "simultaneous" it is meant that two or more requests are valid at the clock edge at which the internal arbiter samples them.

Dual-Port Considerations

For both ports to be operated synchronously, several conditions must be met. The processors must be the same type (Fast or Slow Cycle) as defined by Table 8 and they must have synchronized clocks. Also when processor types are mixed, even though the clocks may be in phase, one frequency may be twice that of the other. So to run both ports synchronous using the status interface, the processors must have related timings (both phase and frequency). If these conditions cannot be met, then one port must run synchronous and the other asynchronous.

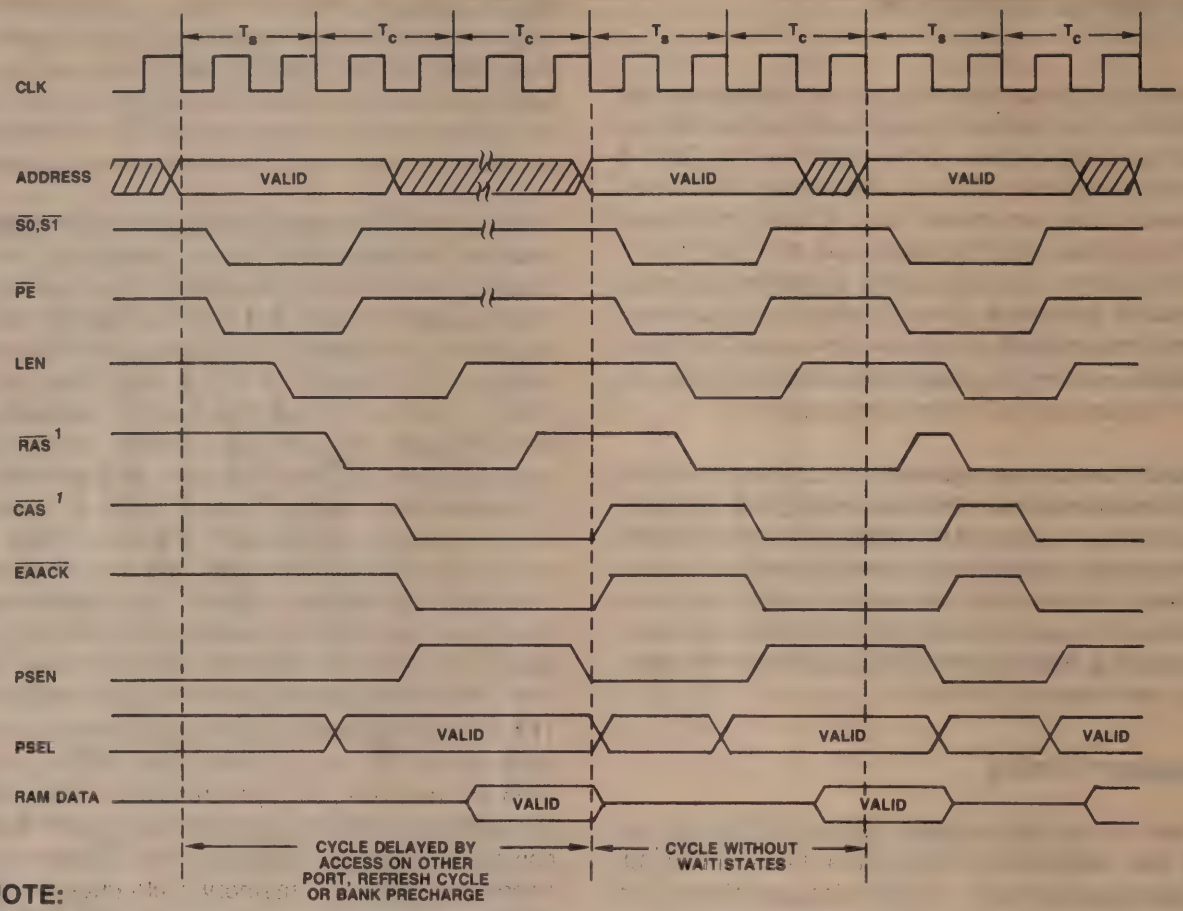
Figure 3 illustrates an example of dual-port operation using the processors in the slow cycle group. Note the use of cross-coupled NAND gates at the MUX output for minimizing contention between the two latches, and the use of flip flops on the status lines of the asynchronous processor for delaying the status and thereby guaranteeing RAS will not be issued, even in the worst case, until address is valid.

Processor Timing

In order to run without wait states, $\overline{AACK}$ must be used and connected to the $\overline{SRDY}$ input of the appropriate bus controller. $\overline{AACK}$ is issued relative to a point within the RAM cycle and has no fixed relationship to the processor's request. The timing is such, however, that the processor will run without wait states, barring refresh cycles, bank precharge, and RAM accesses from the other port. In non-ECC fast cycle, fast RAM, non-extended configurations (80286), $\overline{AACK}$ is issued on the next falling edge of the clock after the edge that issues RAS. In non-ECC, slow cycle, non-extended, or extended with fast RAM cycle configurations (8086, 80188, 80186), $\overline{AACK}$ is issued on the same clock cycle that issues RAS. Figure 14 illustrates the timing relationship between $\overline{AACK}$, the RAM cycle, and the processor cycle for several different situations.

Port Enable ($\overline{PE}$) setup time requirements depend on whether the associated port is configured for synchronous or asynchronous fast or slow cycle operation. In a synchronous fast cycle configuration, $\overline{PE}$ is required to be setup to the same clock edge as the status or commands. If $\overline{PE}$ is true (low), a RAM cycle is started; if not, the cycle is aborted. The memory cycle will only begin when both valid signals ($\overline{PE}$ and $\overline{RD}$ or WR) are recognized at a particular clock edge. In asynchronous operation, $\overline{PE}$ is required to be setup to the same clock edge as the internally synchronized status or commands. Externally, this allows the internal synchronization delay to be added to the status (or command)-to-$\overline{PE}$ delay time, thus allowing for more external decode time that is available in synchronous operation.

The minimum synchronization delay is the additional amount that $\overline{PE}$ must be held valid. If $\overline{PE}$ is not held valid for the maximum synchronization delay time, it is possible that $\overline{PE}$ will go invalid prior to the status or command being synchronized. In such a case the 8207 aborts the cycle. If a memory cycle intended for the 8207 is aborted, then no acknowledge ($\overline{AACK}$ or $\overline{XACK}$) is issued and the processor locks up in endless wait states. Figure 15 illustrates the status (command) timing requirements for synchronous and asynchronous systems. Figures 16 and 17 show a more detailed hook-up of the 8207 to the 8086 and the 80286, respectively.



NOTE:
1. The RAS and CAS shown in figure are different banks being accessed.

Figure 14. iAPX 286/8207 Synchronous-Status Timing Programmed in non-ECC Mode, C0 Configuration (Read Cycle)

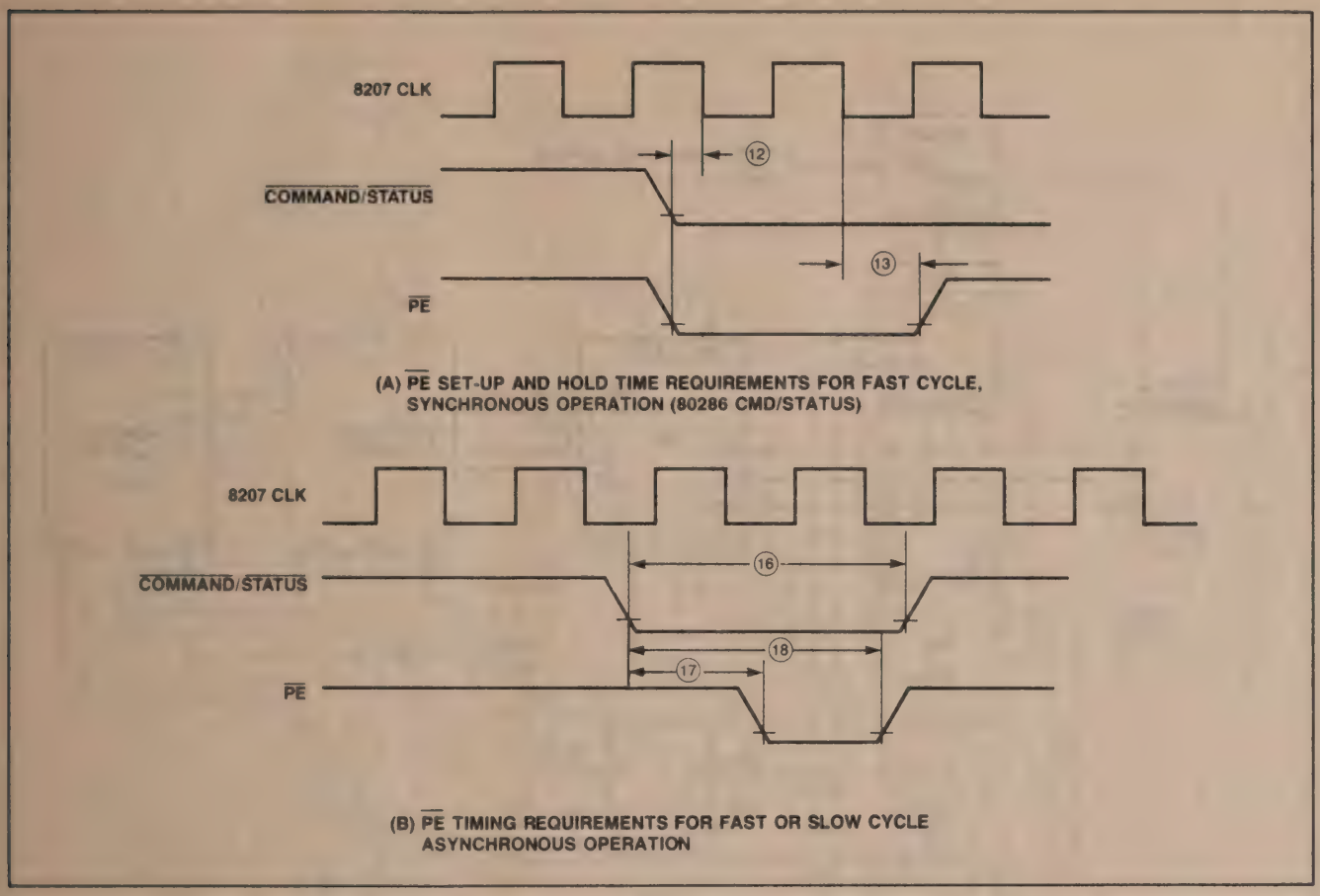


Figure 15.

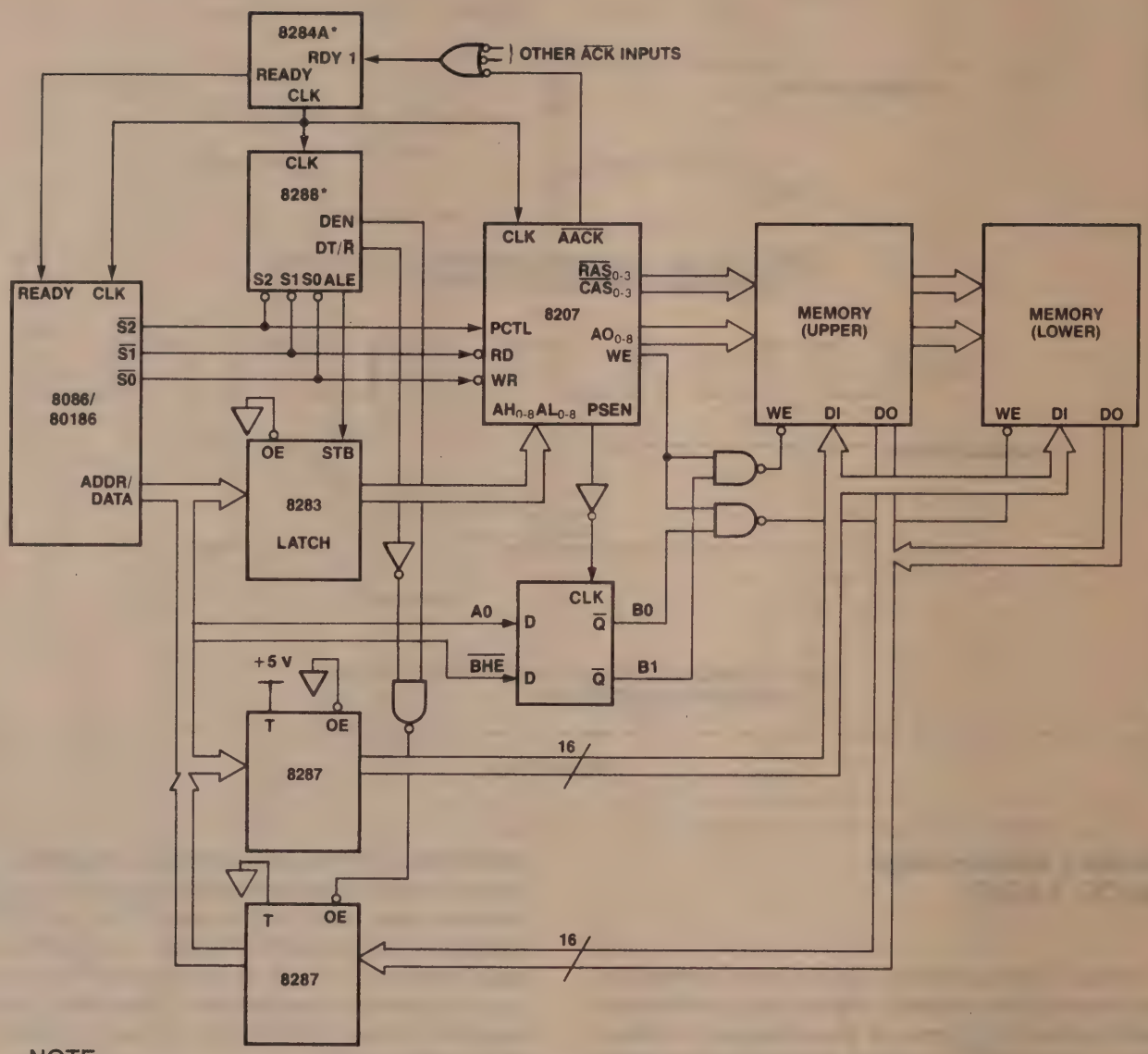
Memory Acknowledge (AACK, XACK)

In system configurations without error correction, two memory acknowledge signals per port are supplied by the 8207. They are the Advanced Acknowledge strobe ($\overline{AACK}$) and the Transfer Acknowledge strobe ($\overline{XACK}$). The CFS programming bit determines for which processor $\overline{AACKA}$ and $\overline{AACKB}$ are optimized, either 80286 (CFS = 1) or 8086/186 (CFS = 0), while the SA and SB programming bits optimize $\overline{AACK}$ for synchronous operation ("early" $\overline{AACK}$) or asynchronous operation ("late" $\overline{AACK}$).

Both the early and late $\overline{AACK}$ strobes are three clocks long for CFS = 1 and two clocks long for CFS = 0. The $\overline{XACK}$ strobe is asserted when data is valid (for reads) or when data may be removed (for writes) and meets the Multibus requirements. $\overline{XACK}$ is removed asynchronously by the command going inactive. Since in asynchronous operation the 8207 removes read data before late $\overline{AACK}$ or $\overline{XACK}$ is recognized by the CPU, the user must provide for data latching in the system until the CPU reads the data. In synchronous operation, data latching is unnecessary since the 8207 will not remove data until the CPU has read it.

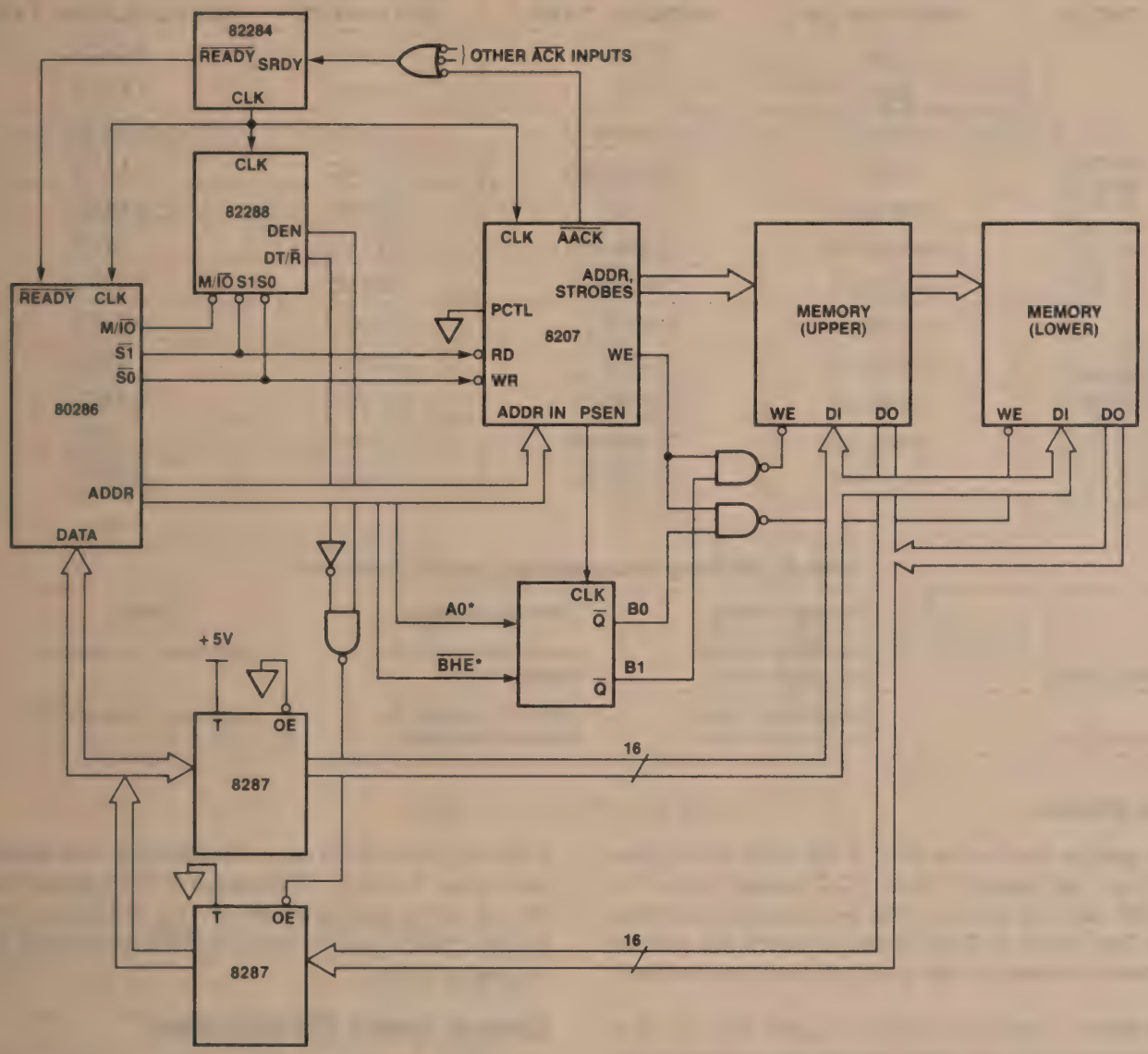
In ECC-based systems there is one memory acknowledge ($\overline{XACK}$ or $\overline{AACK}$) per port and a programming bit associated with each acknowledge. If the X programming bit is active, the strobe is configured as $\overline{XACK}$, while if the bit is inactive, the strobe is configured as $\overline{AACK}$. As in non-ECC, the SA and SB programming bits determine whether the $\overline{AACK}$ strobe is early or late ($\overline{EAACK}$ or $\overline{LAACK}$).

Data will always be valid a fixed time after the occurrence of the advanced acknowledge. Table 9 summarizes the various transfer acknowledge options.



NOTE:
*These components are not necessary when using the 80186. These functions are provided directly by the 80186.

Figure 16. 8086/80186, 8207 Single Port Non-ECC Synchronous Systems



Note: While the 8207 does not need the input addresses latched, A0, $\overline{BHE}$ must come from the latched address bus.

Figure 17. 80286 Hook-up to 8207 Non-ECC Synchronous System–Single Port.

Table 8. Processor Interface/Acknowledge Summary

CYCLE	PROCESSOR	REQUEST TYPE	SYNC/ASYNC INTERFACE	ACKNOWLEDGE TYPE
FAST CYCLE CFS=1	80286	STATUS	SYNC	$\overline{EAACK}$
	80286	STATUS	ASYNC	$\overline{LAACK}$
	80286	COMMAND	SYNC	$\overline{EAACK}$
	80286	COMMAND	ASYNC	$\overline{LAACK}$
	8086/80186	STATUS	ASYNC	$\overline{LAACK}$
	8086/80186	COMMAND	ASYNC	$\overline{LAACK}$
	MULTIBUS	COMMAND	ASYNC	$\overline{XACK}$
SLOW CYCLE CFS=0	8086/80186	STATUS	SYNC	$\overline{EAACK}$
	8086/80186	STATUS	ASYNC	$\overline{LAACK}$
	8086/80186	COMMAND	SYNC	$\overline{EAACK}$
	8086/80186	COMMAND	ASYNC	$\overline{LAACK}$
	MULTIBUS	COMMAND	ASYNC	$\overline{XACK}$

Table 9. Memory Acknowledge Option Summary

	Synchronous	Asynchronous	XACK
Fast Cycle	AACK Optimized for Local 80286	AACK Optimized for Remote 80286	Multibus Compatible
Slow Cycle	AACK Optimized for Local 8086/186	AACK Optimized for Remote 8086/186	Multibus Compatible

Test Modes

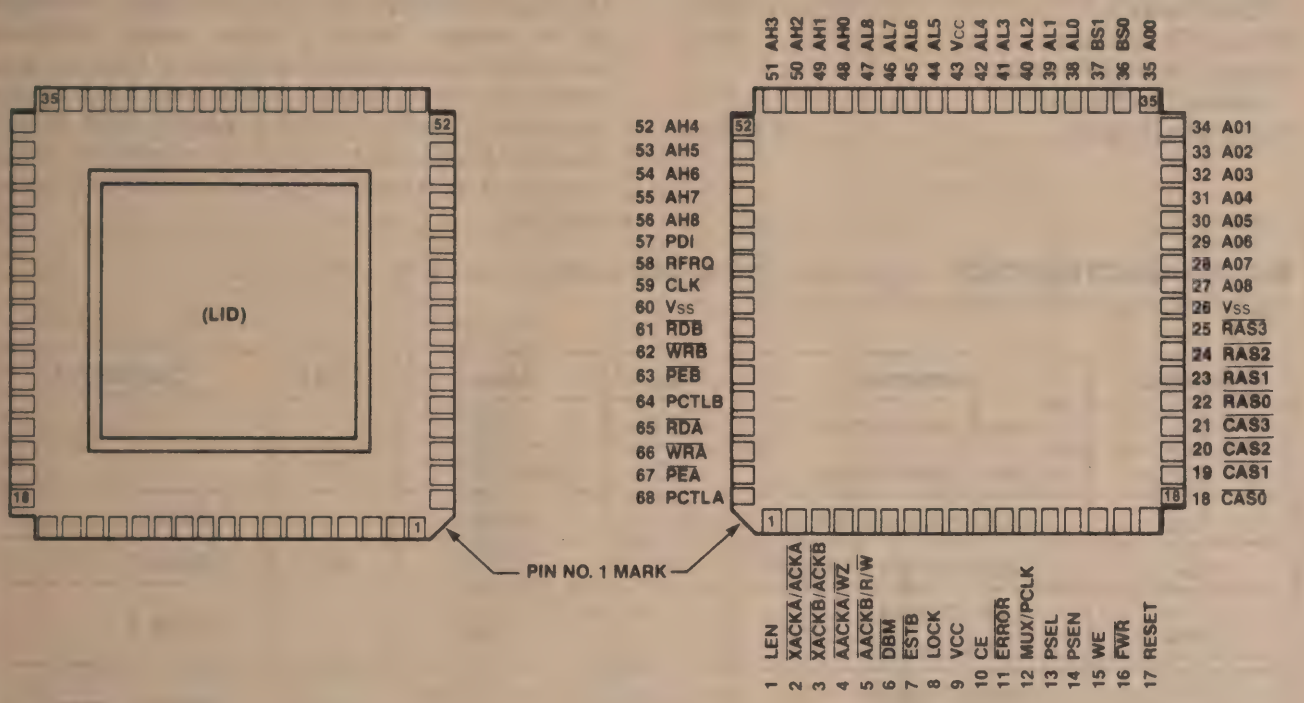
Two special test modes exist in the 8207 to facilitate testing. Test Mode 1 (non-ECC mode) splits the refresh address counter into two separate counters and Test Mode 2 (ECC mode) presets the refresh address counter to a value slightly less than rollover.

Test Mode 1 splits the address counter into two, and increments both counters simultaneously with each refresh address update. By generating external refresh requests, the tester is able to check for proper operation of both counters. Once proper individual counter operation has been established, the 8207 must be returned to normal mode and a second test performed to check that the carry from the first counter increments the second counter. The outputs of the counters are presented-on the address out bus with the same timing as the row and column addresses of a normal scrubbing operation. During Test Mode 1, memory initialization is inhibited, since the 8207, by definition, is in non-ECC mode.

Test Mode 2 sets the internal refresh counter to a value slightly less than rollover. During functional testing other than that covered in Test Mode 1, the 8207 will normally be set in Test Mode 2. Test Mode 2 eliminates memory initialization in ECC mode. This allows quick examination of the circuitry which brings the 8207 out of memory initialization and into normal operation.

General System Considerations

The RAS_{0-3}, CAS_{0-3}, AO_{0-8}, output buffers were designed to directly drive the heavy capacitive loads associated with dynamic RAM arrays. To keep the RAM driver outputs from ringing excessively in the system environment and causing noise in other output pins it is necessary to match the output impedance of the RAM output buffers with the RAM array by using series resistors and to add series resistors to other control outputs for noise reduction if necessary. Each application may have different impedance characteristics and may require different series resistance values. The series resistance values should be determined for each application. In non-ECC systems unused ECC input pins should be tied high or low to improve noise immunity.



NOTE:
LCC is mounted lid-down into socket.

Figure 19. 8207 Pinout Diagram

Packaging

The 8207 is packaged in a 68 lead JEDEC Type A Leadless Chip Carrier (LCC) and in Pin Grid Array (PGA), both in Ceramic. The package designations are R and A respectively, eg:

R 8207-8 LCC, 8 MHz DRAM Controller
A 8207-16 PGA, 16 MHz DRAM Controller

Note: The pin-out of the PGA is the same as the socketed pinout of the LCC.

ABSOLUTE MAXIMUM RATINGS

Ambient Temperature
Under Bias ..... −0° C to +70° C
Storage Temperature ..... −65°C to +150°C
Voltage On Any Pin With
Respect to Ground ..... −.5V to +7V
Power Dissipation ..... 2.5 Watts

NOTICE: Stress above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

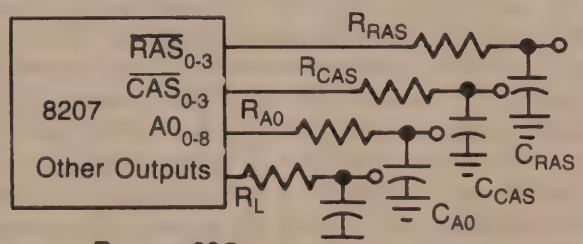
D.C. CHARACTERISTICS

V_{CC} = 5.0V ± 10% for 8207-12, 8207-8, 8207-6; T_A = 0°C to 70°C; V_{SS} = GND V_{CC} = 5.0V ± 5% for 8207-16.

Symbol	Parameter	Min.	Max.	Units	Comments
V_{IL}	Input Low Voltage	−0.5	+0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC} + 0.5	V	
V_{OL}	Output Low Voltage		0.45	V	Note 1
V_{OH}	Output High Voltage	2.4		V	Note 1
V_{ROL}	RAM Output Low Voltage		0.45	V	Note 1
V_{ROH}	RAM Output High Voltage	2.6		V	Note 1
I_{CC}	Supply Current		455	mA	T_A=0°C
I_{LI}	Input Leakage Current		+10	μA	$0V \leq V_{IN} \leq V_{CC}$
V_{CL}	Clock Input Low Voltage	−0.5	+0.6	V	
V_{CH}	Clock Input High Voltage	3.8	V_{CC} + 0.5	V	
C_{IN}	Input Capacitance		20	pF	fc = 1 MHz

NOTE 1:
I_{OL} =5 mA and I_{OH} = -0.2 mA (Typically I_{OL} = 10 mA and I_{OH} = -0.88 mA)
WE : I_{OL} =8 mA

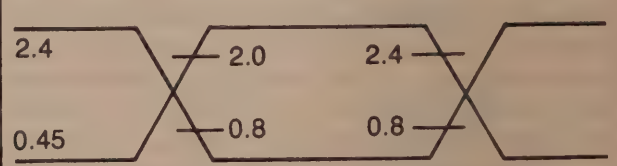
A.C. Testing Load Circuit



R_{RAS} = 39Ω
R_{CAS} = 39Ω
R_{A0} = 22Ω
R_L = 39Ω

C_{RAS} = 150 pF
C_{CAS} = 150 pF
C_{A0} = 380 pF
C_L = 100 pF

A.C. Testing Input, Output Waveform



A.C. Testing inputs (except clock) are driven at 2.4V for a logic "1" and 0.45V for a logic "0" (clock is driven at 4.0V and 0.45V for logic "1" and "0" respectively). Timing measurements are made at 2.0V, 2.4V for logic "1" and 0.8 V for logic "0".

210463-005

A.C. CHARACTERISTICS

V_{CC} = 5V ± 10% for 8207-12, 8207-8, 8207-6; T_A = 0°C to 70°C; V_{CC} = +5V ± 5% for 8207-16.

Measurements made with respect to RAS_{0-3}, CAS_{0-3}, AO_{0-8}, are a +2.4V and 0.8V. All other pins are measured at 2.0V and 0.8V. All times are nsec unless otherwise indicated. Testing done with specified test load.

CLOCK AND PROGRAMMING

Ref.	Symbol	Parameter		8207-16, -8 (FFS=1) Min.	8207-16, -8 (FFS=1) Max.	8207-12, -6 (FFS=0) Min.	8207-12, -6 (FFS=0) Max.	Units	Notes
–	tF	Clock Fall Time			10		10	ns	3
–	tR	Clock Rise Time			10		10	ns	3
1	TCLCL	Clock Period	8207-16	62.5	200			ns	1
			8207-12			83.3	200	ns	1
			8207-8	125	500			ns	2
			8207-6			167	500	ns	2
2	TCL	Clock Low Time	8207-16	15	180			ns	1
			8207-12			20	180	ns	1
			8207-8	TCLCL/2-12				ns	2
			8207-6			TCLCL/2-12		ns	2
3	TCH	Clock High Time	8207-16	20	180			ns	1
			8207-12			25	180	ns	1
			8207-8	TCLCL/3-3				ns	2
			8207-6			TCLCL/3-3		ns	2
4	TRTVCL	Reset to CLK↓ Setup		40		55		ns	4
5	TRTH	Reset Pulse Width		4 TCLCL		4 TCLCL		ns	
6	TPGVRTL	PCTL, PDI, RFRQ to RESET↓ Setup		125		167		ns	5
7	TRTLPGX	PCTL, RFRQ to RESET↓ Hold		10		10		ns	
8	TCLPC	PCLK from CLK↓ Delay			45		55	ns	
9	TPDVCL	PDin to CLK↓ Setup		60		85		ns	
10	TCLPDX	PDin to CLK↓ Hold		40		55		ns	6

RAM WARM-UP AND INITIALIZATION

64	TCLWZL	$\overline{WZ}$ from CLK↓ Delay		40		55	ns	7

SYNCHRONOUS μP PORT INTERFACE

11	TPEVCL	PE to CLK↓ Setup	30		40			2
12	TKVCL	$\overline{RD}$, $\overline{WR}$, $\overline{PE}$, PCTL to CLK↓ Setup	20		25		ns	1
13	TCLKX	$\overline{RD}$, $\overline{WR}$, $\overline{PE}$, PCTL to CLK↓ Hold	0		0		ns	
14	TKVCH	RD, WR, PCTL to CLK↑ Setup	20		30		ns	2

ASYNCHRONOUS μP PORT INTERFACE

15	TRWVCL	$\overline{RD}$, $\overline{WR}$ to CLK↓ Setup	20		30		ns	8, 9
16	TRWL	$\overline{RD}$, $\overline{WR}$ Pulse Width	2TCLCL+30		2TCLCL+40		ns	
17	TRWLPEV	$\overline{PE}$ from $\overline{RD}$, $\overline{WR}$↓ Delay CFS=1		TCLCL-20		TCLCL-30	ns	1
		CFS=0		TCLCL-30		TCLCL-40	ns	2
18	TRWLPEX	$\overline{PE}$to $\overline{RD}$, $\overline{WR}$↓ Hold	2TCLCL+30		2TCLCL+40		ns	
19	TRWLPTV	PCTL from $\overline{RD}$, $\overline{WR}$↓ Delay		TCLCL-30		TCLCL-40	ns	2
20	TRWLPTX	PCTL to $\overline{RD}$, $\overline{WR}$↓ Hold	2TCLCL+30		2TCLCL+40		ns	2
21	TRWLPTV	PCTL from $\overline{RD}$, $\overline{WR}$↓ Delay		2TCLCL-20		2TCLCL-30	ns	1
22	TRWLPTX	PCTL to $\overline{RD}$, $\overline{WR}$↓ Hold	3TCLCL+30		3TCLCL+40		ns	1

210463-005

A.C. CHARACTERISTICS (Continued)

RAM INTERFACE

Ref.	Symbol	Parameter		8207-16, -8 (FFS=1) Min.	8207-16, -8 (FFS=1) Max.	8207-12, -6 (FFS=0) Min.	8207-12, -6 (FFS=0) Max.	Units	Notes
23	TAVCL	AL, AH, BS to CLK↓ Setup		35+tASR		45+tASR		ns	10
24	TCLAX	AL, AH, BS to CLK↓ Hold		0		0		ns	
25	TCLLN	LEN from CLK↓ Delay			35		45	ns	
26	TCLRSL	RAS↓ from CLK↓ Delay			35		45	ns	
27	TRCD	$\overline{RAS}$ to $\overline{CAS}$ Delay	CFS=1 CFS=0 CFS=0	TCLCL-25 TCLCL/2-25 75		TCLCL-30 TCLCL/2-30 70		ns ns ns	1, 14 11, 14 12, 14
28	TCLRSH	RAS↑ from CLK↓ Delay			50		60	ns	
29	TRAH	Row Ao to $\overline{RAS}$ Hold	CFS=1 CFS=0 CFS=0	TCLCL/2-10 TCLCL/4-10 40		TCLCL/2-15 TCLCL/4-15 35		ns ns ns	1, 13, 15 11, 15 12, 15
30	TASR	Row A0 to $\overline{RAS}$ setup							10, 18
31	TASC	Column A0 to $\overline{CAS}$↓ Setup	CFS=1 CFS=0	0 5		5 5		ns ns	13, 19, 20 13, 19, 20
32	TCAH	Column A0 to $\overline{CAS}$ Hold		(See DRAM Interface Tables)					21
33	TCLCSL	$\overline{CAS}$↓ from CLK↓ Delay		TCLCL/4 +30	TCLCL/1.8 +53	TCLCL/4 +30	TCLCL/1.8 +72	ns ns	11 12
34	TCLCSL	$\overline{CAS}$↓ from CLK↓ Delay			35		40	ns	1
35	TCLCSH	$\overline{CAS}$↑ from CLK↓ Delay			50		60	ns	
36	TCLW	WE from CLK↓ Delay			35		45	ns	
37	TCLTKL	$\overline{XACK}$↓ from CLK↓ Delay			35		45	ns	
38	TRWLTKH	$\overline{XACK}$↑ from $\overline{RD}$↑, $\overline{WR}$↑ Delay			50		55	ns	
39	TCLAKL	$\overline{AACK}$↓ from $\overline{CLK}$↓ Delay			35		45	ns	
40	TCLAKH	$\overline{AACK}$↑ from CLK↓ Delay			50		60	ns	
41	TCLDL	$\overline{DBM}$ from CLK↓ Delay			35		45	ns	

ECC INTERFACE

42	TWRLFV	$\overline{FWR}$ from $\overline{WR}$↓ Delay	CFS=1 CFS=0		2TCLCL-40 TCLCL+ TCL-40		2TCLCL-50 TCLCL+ TCL-65	ns ns	1, 22 2, 22
43	TFVCL	$\overline{FWR}$ to CLK↓ Setup		40		50		ns	23
44	TCLFX	$\overline{FWR}$ to CLK↓ Hold		0		0		ns	24
45	TEVCL	$\overline{ERROR}$ to CLK↓ Setup		20		25		ns	25, 26
46	TCLEX	$\overline{ERROR}$ to CLK↓ Hold		0		0		ns	
47	TCLRL	R/$\overline{W}$ from CLK↓ Delay			40		45	ns	
48	TCLRH	R/$\overline{W}$ from CLK↓ Delay			50		60	ns	
49	TCEVCL	CE to CLK↓ Setup		20		25		ns	25, 27
50	TCLCEX	CE to CLK↓ Hold		0		0		ns	
51	TCLES	$\overline{ESTB}$ from CLK↓ Delay			35		45	ns	

210463-005

A.C. CHARACTERISTICS (Continued)

PORT SWITCHING AND LOCK

Ref.	Symbol	Parameter	8207-16, -8 (FFS=1) Min.	8207-16, -8 (FFS=1) Max.	8207-12, -6 (FFS=0) Min.	8207-12, -6 (FFS=0) Max.	Units	Notes
52	TCLMV	MUX from CLK↓ Delay		45		55	ns	
53	TCLPNV	PSEN from CLK↓ Delay	TCL TCL	60 TCL+35	TCL TCL	60 TCL+35	ns ns	28 29
54	TCLPSV	PSEL from CLK↓		35		45	ns	
55	TLKVCL	LOCK to CLK↓ Setup	30		40		ns	30, 31
56	TCLLKX	LOCK to CLK↓ Hold	10		10		ns	30, 31
57	TRWLLKV	LOCK from RD↓, WR↓ Delay		2TCLCL-30		2TCLCL-40	ns	31, 32
58	TRWHLKX	LOCK to RD↑, WR↑ Hold	3TCLCL+30		3TCLCL+40		ns	31, 32

REFRESH REQUEST

59	TRFVCL	RFRQ to CLK↓ Setup	20		25		ns	
60	TCLRFX	RFRQ to CLK↓ Hold	10		10		ns	
61	TFRFH	Failsafe RFRQ Pulse Width	TCLCL+30		TCLCL+40		ns	33
62	TRFXCL	Single RFRQ Inactive to CLK↓ Setup	20		30		ns	34
63	TBRFH	Burst RFRQ Pulse Width	2TCLCL+30		2TCLCL+40		ns	33

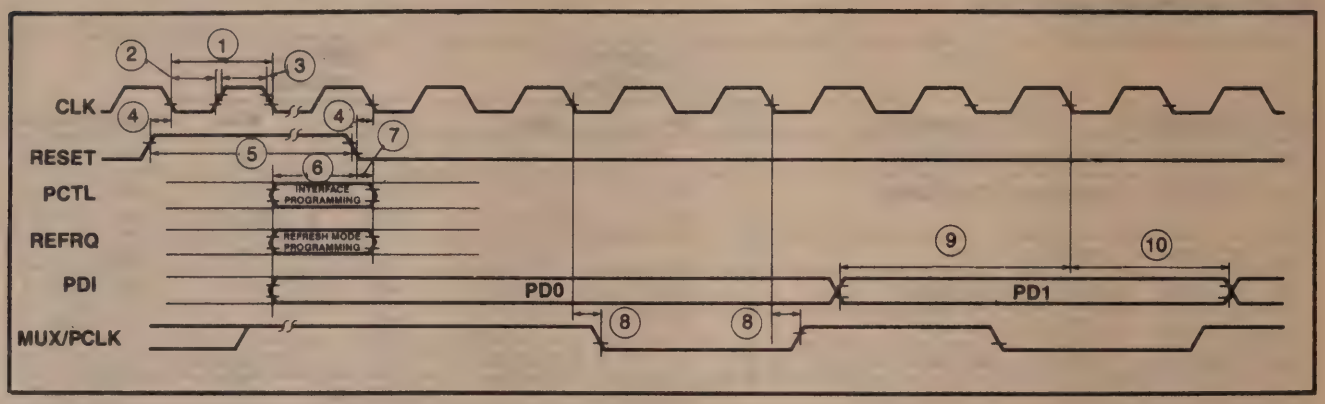
NOTES:

1. Specification when programmed in the Fast Cycle processor mode (iAPX 286 mode).
2. Specification when programmed in the Slow Cycle processor mode (iAPX 186 mode).
3. tR and tF are referenced from the 3.5V and 1.0V levels.
4. RESET is internally synchronized to CLK. Hence a set-up time is required only to guarantee its recognition at a particular clock edge.
5. The first programming bit (PD0) is also sampled by RESET going low.
6. TCLPDX is guaranteed if programming data is shifted using PCLK.
7. WZ is issued only in ECC mode
8. TRWVCL is not required for an asynchronous command except to guarantee its recognition at a particular clock edge.
9. Valid when programmed in either Fast or Slow Cycle mode.
10. tASR is a user specified parameter and its value should be added accordingly to TAVCL.
11. When programmed in Slow Cycle mode and 125 ns ≤ TCLCL < 200 ns.
12. When programmed in Slow Cycle mode and 200 ns ≤ TCLCL.
13. Specification for Test Load conditions.
14. tRCD (actual) = tRCD (specification) +0.06 (ΔC_{RAS}) - 0.06 (ΔC_{CAS}) where ΔC = C (test load) - C (actual) in pF. (These are first order approximations.)
15. tRAH (actual) = tRAH (specification) + 0.06 (ΔC_{RAS}) - 0.022 (ΔC_{A0}) where ΔC = C (test load) - C (actual) in pF. (These are first order approximations.)
16.
17.
18. tASR (actual) = tASR (specification) +0.06 (ΔC_{A0}) - 0.025 (ΔC_{RAS}) where ΔC = C (test load) - C (actual) in pF. (These are first order approximations.)
19. tASC (actual) = tASC (specification) +0.06 (ΔC_{A0}) - 0.025 (ΔC_{CAS}) where ΔC = C (test load) - C (actual) in pF. (These are first order approximations.)
20. tASC is a function of clock frequency and thus varies with changes in frequency. A mimimum value is specified.
21. See 8207 DRAM Interface Tables 14 - 18.
22. TWRLFV is defined for both synchronous and asynchronous $\overline{FWR}$. In systems in which $\overline{FWR}$ is decoded directly from the address inputs to the 8207. TCLFV is automatically guaranteed by TCLAV.
23. TFVCL is defined for synchronous $\overline{FWR}$.
24. TCLFV is defined for both synchronous and asynchronous $\overline{FWR}$. In systems in which $\overline{FWR}$ is decoded directly from the address inputs to the 8207. TCLFV is automatically guaranteed by TCLAV.
25. $\overline{ERROR}$ and CE are set-up to CLK↓ in fast cycle mode and CLK↑ in slow cycle mode.
26. $\overline{ERROR}$ is set-up to the same edge as R/$\overline{W}$ is referenced to, in RMW cycles.
27. CE is set-up to the same edge as WE is referenced to in RMW cycles.
28. Specification when TCL < 25 ns.
29. Specification when TCL ≥ 25 ns.
30. Synchronous operation only. Must arrive by the second clock falling edge after the clock edge which recognizes the command in order to be effective.
31. LOCK must be held active for the entire period the opposite port must be locked out. One clock after the release of LOCK the opposite port will be able to obtain access to memory.
32. Asynchronous mode only. In this mode a synchronizer stage is used internally in the 8207 to synchronize up LOCK. TRWLLKV and TRWHLKX are only required for guaranteeing that LOCK will be recognized for the requesting port, but these parameters are not required for correct 8207 operation.
33. TFRFH and TBRFH pertain to asynchronous operation only.
34. Single RFRQ cannot by supplied asynchronously.

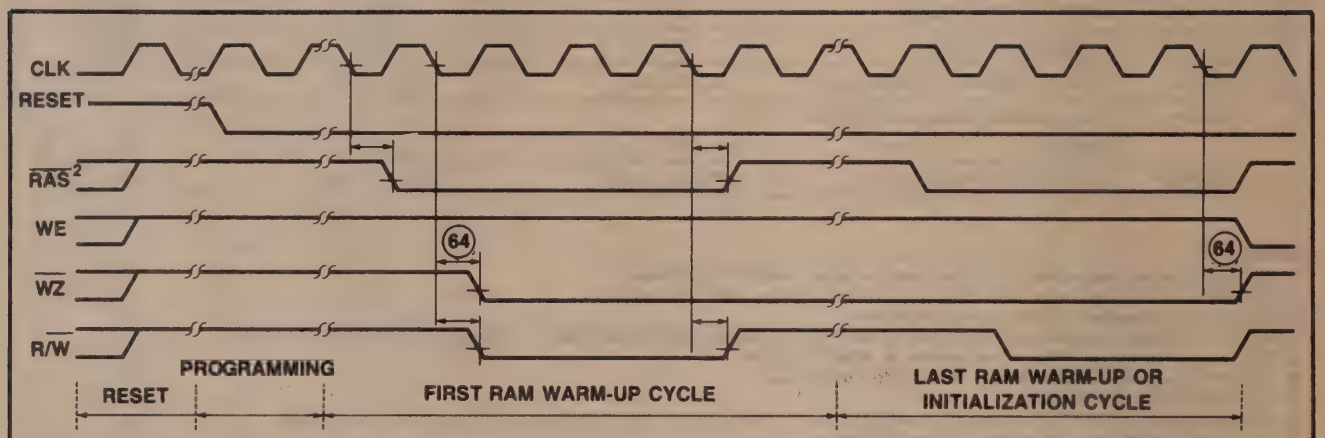
210463-005

WAVEFORMS

Clock and Programming Timings



RAM Warm-up and Memory Initialization Cycles



NOTES:

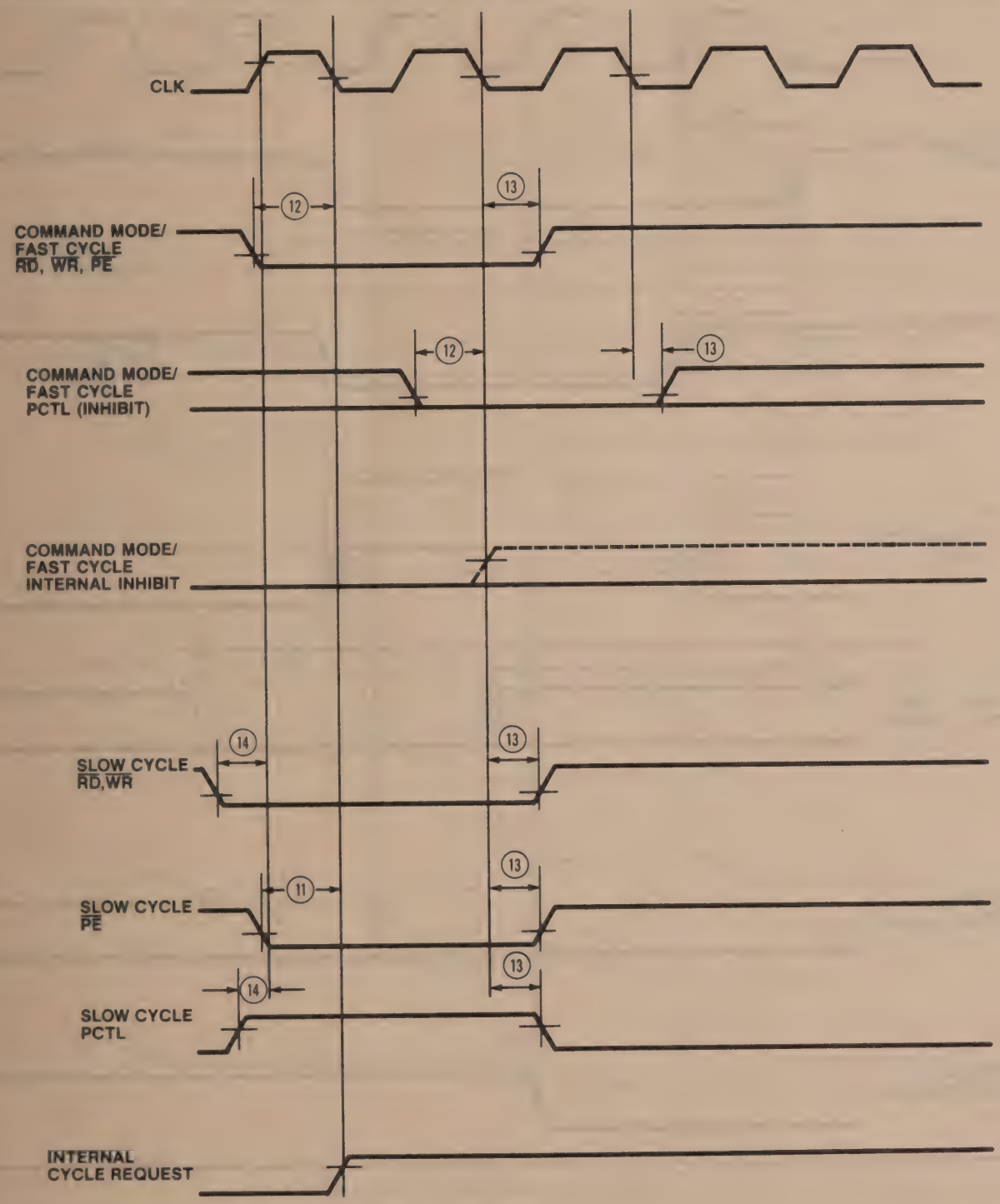
1. When in non-ECC mode or in ECC mode with the TM2 programming bit on, there are no initialization cycles, when in ECC mode with TM2 off, the dummy cycles are followed by initialization cycles.

2. The present example assumes a RAS four clocks long.

210463-005

WAVEFORMS (Continued)

Synchronous Port Interface

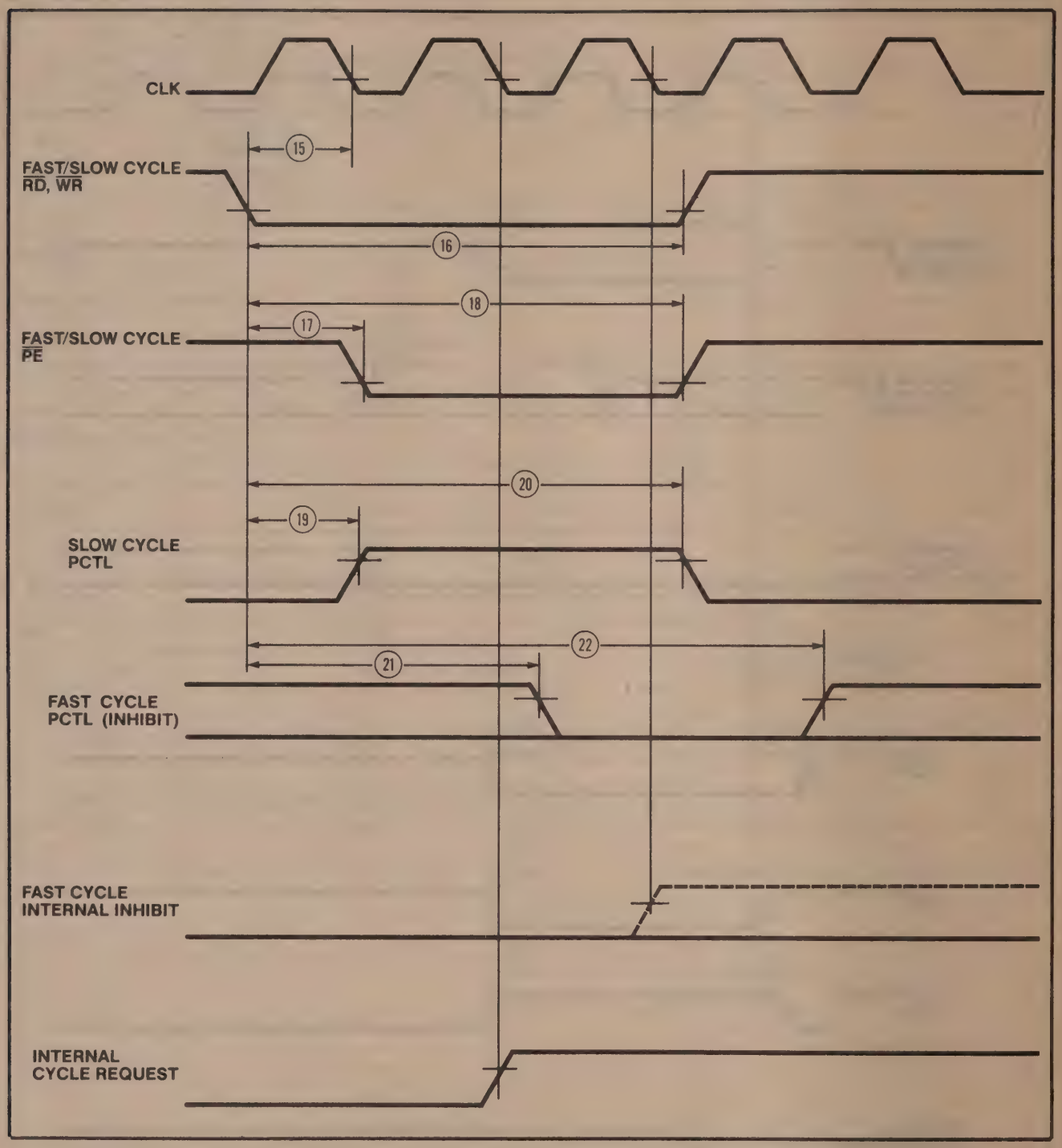


NOTE:
Actual transitions are programmable. Refer to Tables 12 and 13.

210463-005

WAVEFORMS (Continued)

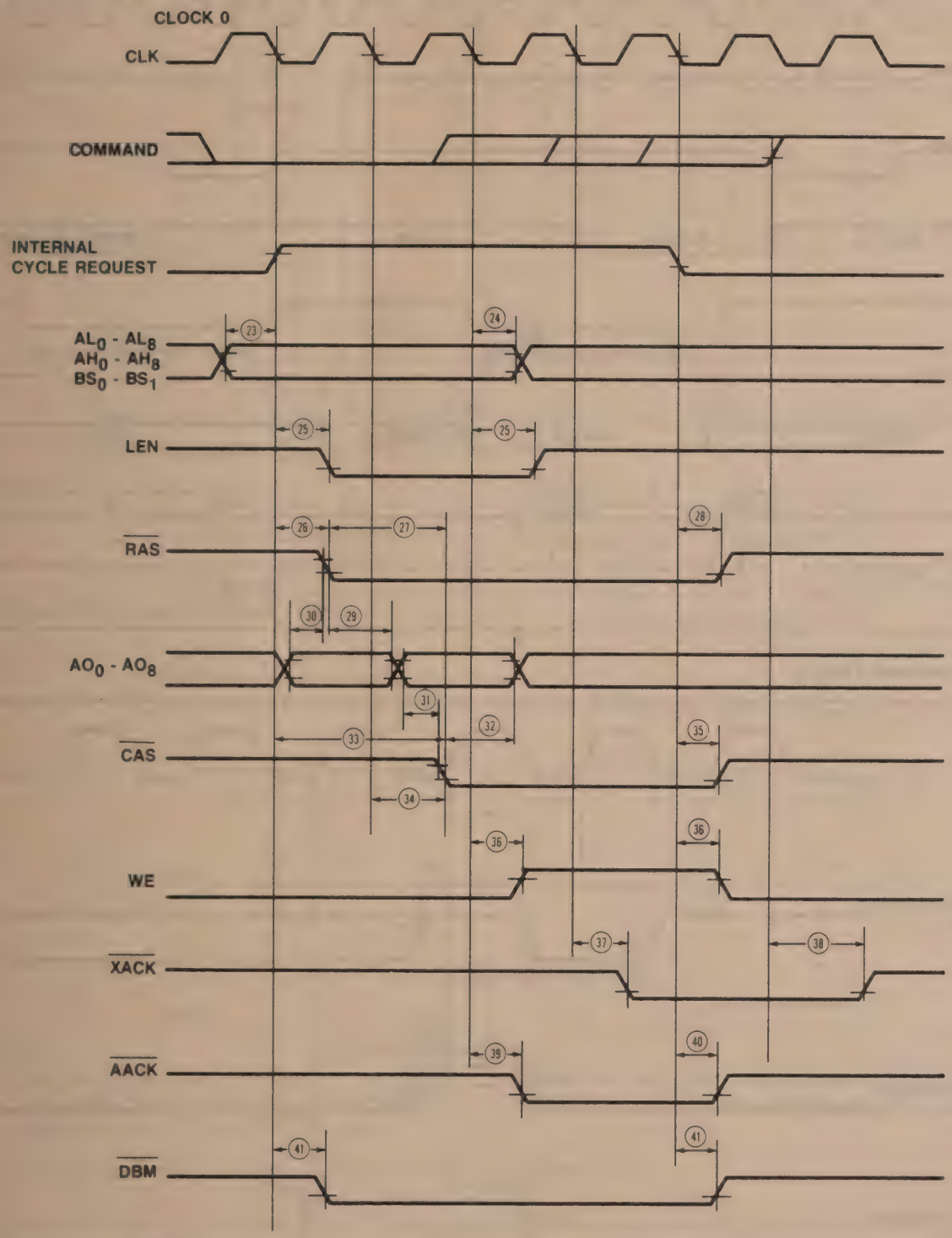
Asynchronous Port Interface



210463-005

WAVEFORMS (Continued)

RAM Interface Timing
ECC and Non-ECC Mode

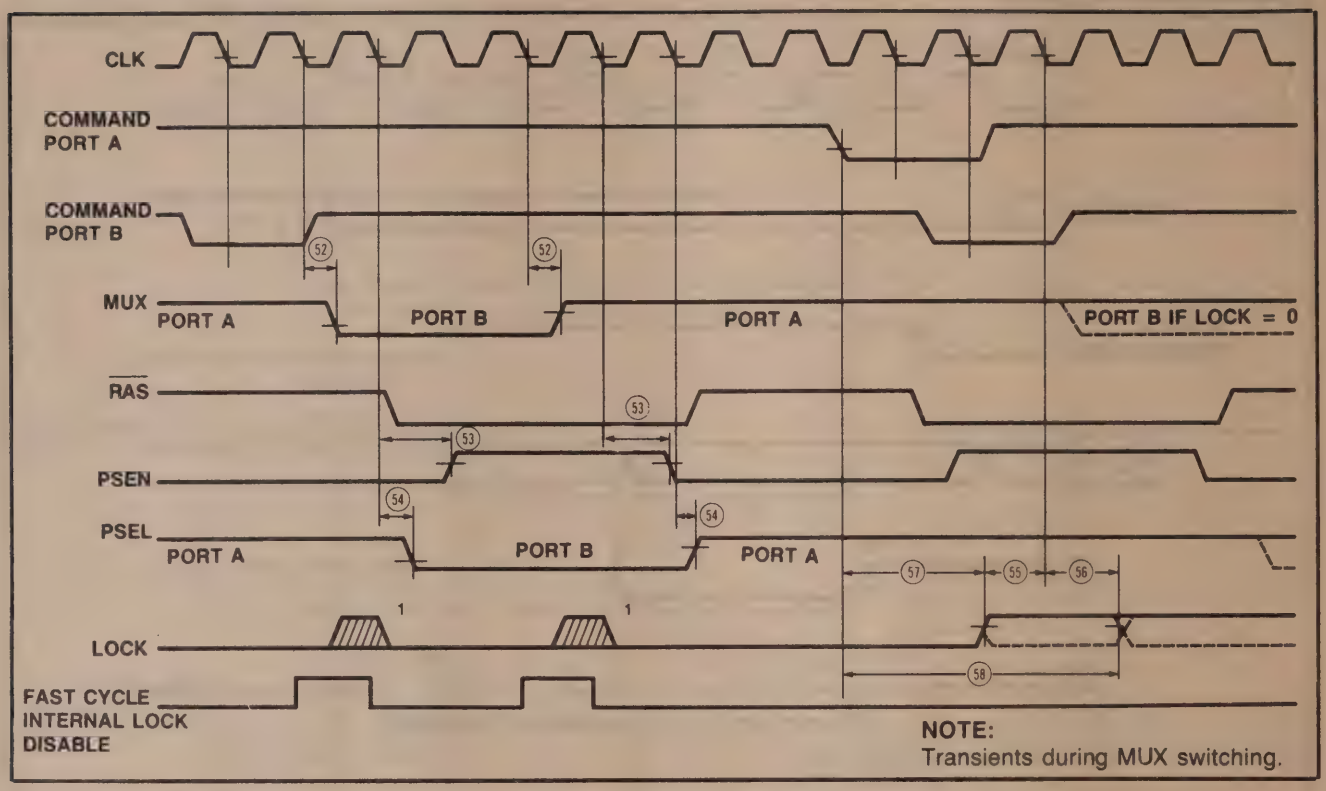


NOTE:
Actual transitions are programmable. Refer to Tables 12 and 13.

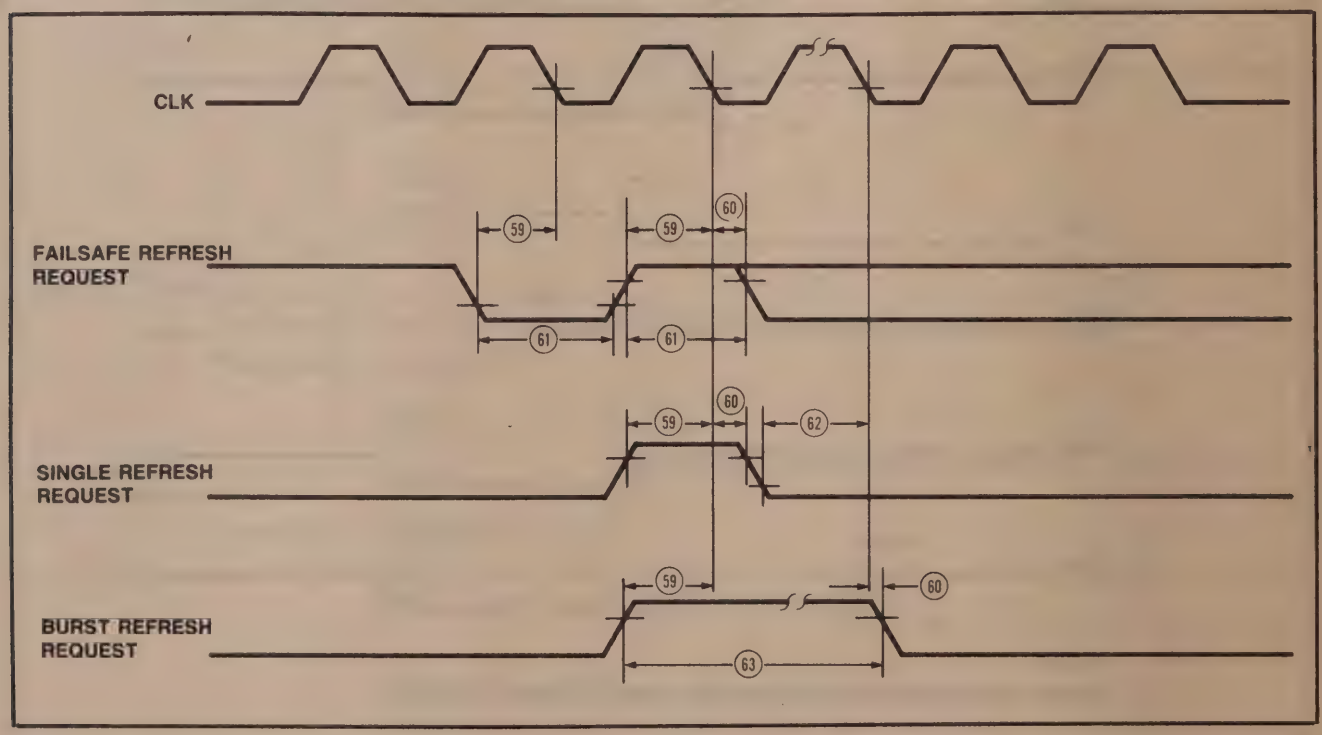
210463-005

WAVEFORMS (Continued)

Port Switching and Lock Timing



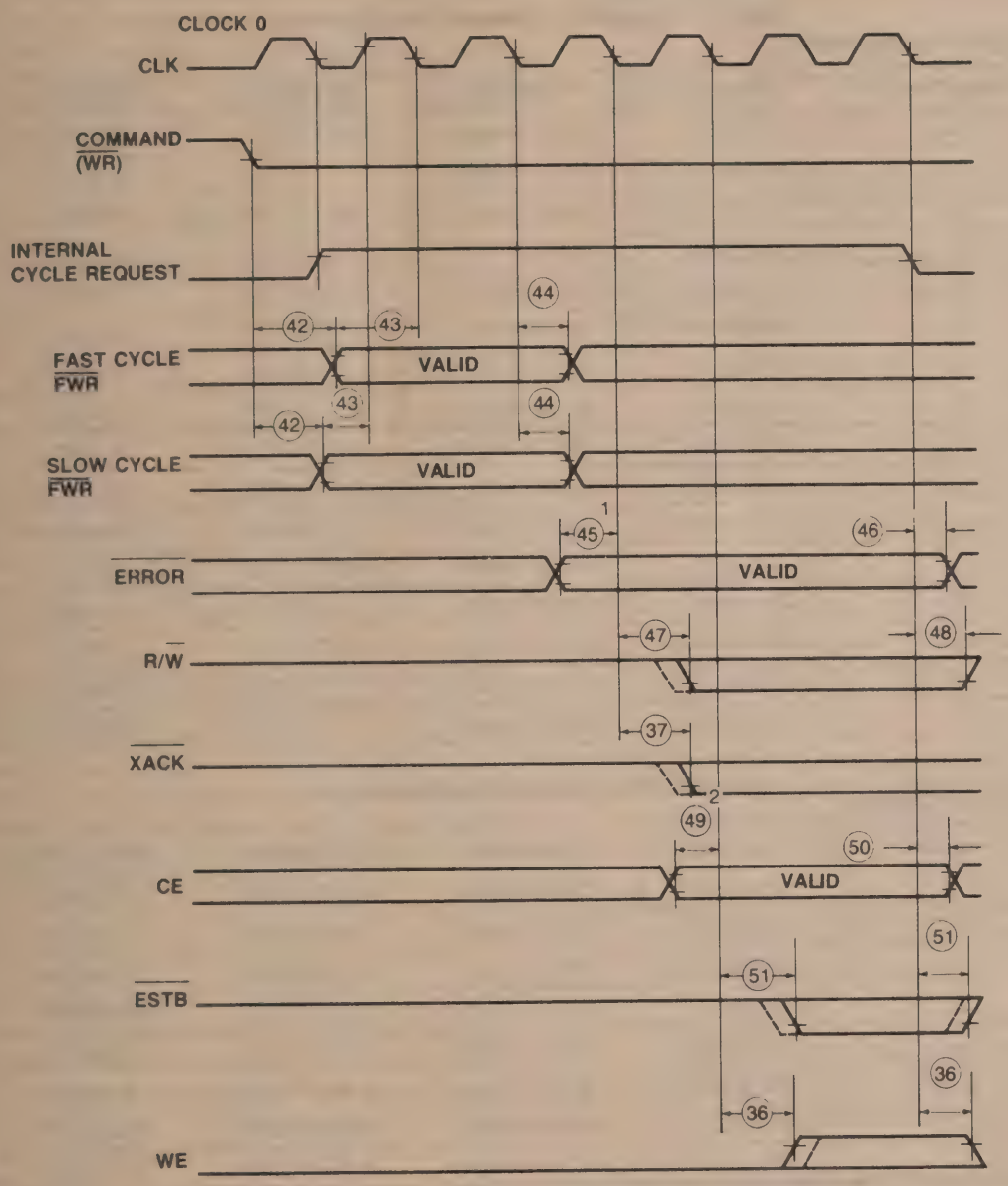
Refresh Request Timing



210463-005

WAVEFORMS (Continued)

ECC Interface Timing



NOTE:

1. This parameter is set-up to the falling edge of clock, as shown, for fast cycle configurations. It is set-up to the rising edge of clock if in slow cycle configurations. Table 13A shows which clock and clock edge these signals are set-up in the R/$\overline{W}$ L column.

2. CE is set-up to the same edge as WE is referenced to in RMW cycles.

210463-005

Table 13 A. Timing Chart — ECC Mode

		PSEN		PSEL		$\overline{DBM}$		$\overline{LEN}$		$\overline{RAS}$		$\overline{CAS}$		R/$\overline{W}$		WE	
C_n	CYCLE	H	L	V	$\overline{V}$	L	H	L	H	L	H	L	H	L	H	H	L
C_0	RD, RF	0↓	5↓	0↓	6↓	0↓	6↓	0↓	2↓	0↓	4↓	1↓	6↓				
	WR	0↓	5↓	0↓	6↓			0↓	2↓	0↓	6↓	1↓	6↓	1↓	6↓	3↓	6↓
	RMW	0↓	8↓	0↓	9↓	0↓	9↓	0↓	2↓	0↓	9↓	1↓	9↓	4↓	9↓	6↓	9↓
C_1	RD, RF	0↓	5↓	0↓	6↓	0↓	6↓	0↓	2↓	0↓	4↓	1↓	6↓				
	WR	0↓	5↓	0↓	6↓			0↓	2↓	0↓	6↓	1↓	6↓	1↓	6↓	3↓	6↓
	RMW	0↓	8↓	0↓	9↓	0↓	9↓	0↓	2↓	0↓	9↓	1↓	9↓	4↓	9↓	6↓	9↓
C_2	RD, RF	0↓	6↓	0↓	7↓	0↓	7↓	0↓	2↓	0↓	5↓	1↓	7↓				
	WR	0↓	6↓	0↓	7↓			0↓	2↓	0↓	7↓	1↓	7↓	1↓	7↓	4↓	7↓
	RMW	0↓	10↓	0↓	11↓	0↓	11↓	0↓	2↓	0↓	11↓	1↓	11↓	5↓	11↓	8↓	11↓
C_3	RD, RF	0↓	6↓	0↓	7↓	0↓	7↓	0↓	2↓	0↓	5↓	1↓	7↓				
	WR	0↓	6↓	0↓	7↓			0↓	2↓	0↓	7↓	1↓	7↓	1↓	7↓	4↓	7↓
	RMW	0↓	10↓	0↓	11↓	0↓	11↓	0↓	2↓	0↓	11↓	1↓	11↓	5↓	11↓	8↓	11↓
C_4	RD, RF	0↓	3↓	0↓	4↓	0↓	4↓	0↓	2↓	0↓	3↓	0↓	4↓				
	WR	0↓	4↓	0↓	5↓			0↓	2↓	0↓	5↓	0↓	5↓	1↑	5↓	3↑	5↓
	RMW	0↓	6↓	0↓	7↓	0↓	7↓	0↓	2↓	0↓	7↓	0↓	7↓	3↑	7↓	5↑	7↓
C_5	RD, RF	0↓	3↓	0↓	4↓	0↓	4↓	0↓	2↓	0↓	3↓	0↓	4↓				
	WR	0↓	4↓	0↓	5↓			0↓	2↓	0↓	5↓	0↓	5↓	1↑	5↓	3↑	5↓
	RMW	0↓	6↓	0↓	7↓	0↓	7↓	0↓	2↓	0↓	7↓	0↓	7↓	3↑	7↓	5↑	7↓
C_6	RD, RF	0↓	3↓	0↓	4↓	0↓	4↓	0↓	2↓	0↓	3↓	0↓	4↓				
	WR	0↓	3↓	0↓	4↓			0↓	2↓	0↓	4↓	0↓	4↓	1↑	4↓	2↑	4↓
	RMW	0↓	4↓	0↓	5↓	0↓	5↓	0↓	2↓	0↓	5↓	0↓	5↓	2↑	5↓	3↑	5↓

210463-005

Table 13 B. Timing Chart — ECC Mode

		COL ADDR		$\overline{ESTB}$		$\overline{EAACK}$		$\overline{LAACK}$		$\overline{XACK}$		MUX	
C_n	CYCLE	V	V	L	H	L	H	L	H	L	H	V	V
C_0	RD, RF	0↓	2↓			2↓	5↓	3↓	6↓	4↓	RD	−2↓	2↓
	WR	0↓	2↓			2↓	5↓	2↓	5↓	4↓	WR	−2↓	2↓
	RMW	0↓	2↓	6↓	8↓	5↓	8↓	5↓	8↓	7↓	WR	−2↓	2↓
C_1	RD, RF	0↓	3↓			3↓	6↓	3↓	6↓	4↓	RD	−2↓	2↓
	WR	0↓	3↓			2↓	5↓	2↓	5↓	4↓	WR	−2↓	2↓
	RMW	0↓	3↓	6↓	8↓	5↓	8↓	5↓	8↓	7↓	WR	−2↓	2↓
C_2	RD, RF	0↓	3↓			4↓	7↓	4↓	7↓	5↓	RD	−2↓	2↓
	WR	0↓	3↓			3↓	6↓	3↓	6↓	5↓	WR	−2↓	2↓
	RMW	0↓	3↓	8↓	10↓	7↓	10↓	7↓	10↓	9↓	WR	−2↓	2↓
C_3	RD, RF	0↓	3↓			4↓	7↓	5↓	8↓	5↓	RD	−2↓	2↓
	WR	0↓	3↓			3↓	6↓	3↓	6↓	5↓	WR	−2↓	2↓
	RMW	0↓	3↓	8↓	10↓	7↓	10↓	7↓	10↓	9↓	WR	−2↓	2↓
C_4	RD, RF	0↓	2↓			1↓	3↓	2↑	4↑	3↑	RD	−1↓	2↓
	WR	0↓	2↓			1↓	3↓	2↑	4↑	3↓	WR	−1↓	2↓
	RMW	0↓	2↓	5↑	6↑	3↓	5↓	4↑	6↑	5↓	WR	−1↓	2↓
C_5	RD, RF	0↓	2↓			2↓	4↓	3↑	5↑	3↑	RD	−1↓	2↓
	WR	0↓	2↓			1↓	3↓	2↑	4↑	3↓	WR	−1↓	2↓
	RMW	0↓	2↓	5↑	6↑	3↓	5↓	4↑	6↑	5↓	WR	−1↓	2↓
C_6	RD, RF	0↓	2↓			1↓	3↓	1↑	3↑	2↑	RD	−1↓	2↓
	WR	0↓	2↓			1↓	3↓	1↑	3↑	2↓	WR	−1↓	2↓
	RMW	0↓	2↓	3↑	4↑	1↓	3↓	2↑	4↑	3↓	WR	−1↓	2↓

210463-005

Table 15. Non-ECC Mode – WR Cycle

Parameter	Fast Cycle Configurations			Slow Cycle Configurations		Notes
	C_0	C_1	C_2	C_3	C_4	
tRP	3TCLCL−T26	3TCLCL−T26	3TCLCL−T26	2TCLCL−T26	2TCLCL−T26	1
tCPN	4TCLCL−T35	4TCLCL−T35	4TCLCL−T35	2.5TCLCL−T35	2.5TCLCL−T35	1
tRSH	4TCLCL−T34	4TCLCL−T34	4TCLCL−T34	4TCLCL−T34	4TCLCL−T34	1
tCSH	5TCLCL−T26	5TCLCL−T26	5TCLCL−T26	4TCLCL−T26	4TCLCL−T26	1
tCAH	TCLCL−T34	2TCLCL−T34	2TCLCL−T34	2TCLCL−T34	2TCLCL−T34	1
tAR	2TCLCL−T26	3TCLCL−T26	3TCLCL−T26	2TCLCL−T26	2TCLCL−T26	1
tT	3/30	3/30	3/30	3/30	3/30	2
tRWC	8TCLCL	8TCLCL	8TCLCL	6TCLCL	6TCLCL	1
tRRW	5TCLCL−T26	5TCLCL−T26	5TCLCL−T26	4TCLCL−T26	4TCLCL−T26	1
tCRW	4TCLCL−T34	4TCLCL−T34	4TCLCL−T34	4TCLCL−T34	4TCLCL−T34	1
tWCH	3TCLCL+TCL −T34	3TCLCL+TCL −T34	3TCLCL+TCL −T34	3TCLCL+TCL −T34	3TCLCL+TCL −T34	1, 3
tWCR	4TCLCL+TCL −T26	4TCLCL+TCL −T26	4TCLCL+TCL −T26	3TCLCL+TCL −T26	3TCLCL+TCL −T26	1, 3
tWP	2TCLCL+TCL −T36−TBUF	2TCLCL+TCL −T36−TBUF	2TCLCL+TCL −T36−TBUF	2TCLCL−T36 −TBUF	2TCLCL−T36 −TBUF	1
tRWL	3TCLCL−T36 −TBUF	3TCLCL−T36 −TBUF	3TCLCL−T36 −TBUF	3TCLCL−TCL −T36−TBUF	3TCLCL−TCL −T36−TBUF	1
tCWL	3TCLCL−T36 −TBUF	3TCLCL−T36 −TBUF	3TCLCL−T36 −TBUF	3TCLCL−TCL −T36−TBUF	3TCLCL−TCL −T36−TBUF	1

210463-005

Table 16 A. ECC Mode — RD, RF Cycles

Parameter	Fast Cycle Mode				
	C_0	C_1	C_2	C_3	Notes
tRP	4TCLCL−T26	4TCLCL−T26	4TCLCL−T26	4TCLCL−T26	1
tCPN	3TCLCL−T35	3TCLCL−T35	3TCLCL−T35	3TCLCL−T35	1
tRSH	3TCLCL−T34	3TCLCL−T34	4TCLCL−T34	4TCLCL−T34	1
tCSH	6TCLCL−T26	6TCLCL−T26	7TCLCL−T26	7TCLCL−T26	1
tCAH	TCLCL−T34	2TCLCL−T34	2TCLCL−T34	2TCLCL−T34	1
tAR	2TCLCL−T26	3TCLCL−T26	3TCLCL−T26	3TCLCL−T26	1
tT	3/30	3/30	3/30	3/30	2
tRC	8TCLCL	8TCLCL	9TCLCL	9TCLCL	1
tRAS	4TCLCL−T26	4TCLCL−T26	5TCLCL−T26	5TCLCL−T26	1
tCAS	5TCLCL−T34	5TCLCL−T34	6TCLCL−T34	6TCLCL−T34	1
tRCS	TCLCL−T36 −TBUF	TCLCL−T36 −TBUF	TCLCL−T36 −TBUF	TCLCL−T36 −TBUF	1

Table 16 B. ECC Mode — RD, RF Cycles

Parameter	Slow Cycle Mode			
	C_4	C_5	C_6	Notes
tRP	2TCLCL−T26	2TCLCL−T26	2TCLCL−T26	1
tCPN	1.5TCLCL−T35	1.5TCLCL−T35	1.5TCLCL−T35	1
tRSH	3TCLCL−T34	3TCLCL−T34	3TCLCL−T34	1
tCSH	4TCLCL−T26	4TCLCL−T26	4TCLCL−T26	1
tCAH	2TCLCL−T34	2TCLCL−T34	2TCLCL−T34	1
tAR	2TCLCL−T26	2TCLCL−T26	2TCLCL−T26	1
tT	3/30	3/30	3/30	2
tRC	5TCLCL	5TCLCL	5TCLCL	1
tRAS	3TCLCL−T26	3TCLCL−T26	3TCLCL−T26	1
tCAS	4TCLCL−T34	4TCLCL−T34	4TCLCL−T34	1
tRCS	0.5TCLCL−T36 −TBUF	0.5TCLCL−T36 −TBUF	0.5TCLCL−T36 −TBUF	1

210463-005

Table 18 A. ECC Mode — RMW

Parameters	Fast Cycle Mode				
	C_0	C_1	C_2	C_3	Notes
tRP	3TCLCL−T26	3TCLCL−T26	3TCLCL−T26	3TCLCL−T26	1
tCPN	4TCLCL−T35	4TCLCL−T35	4TCLCL−T35	4TCLCL−T35	1
tRSH	8TCLCL−T34	8TCLCL−T34	10TCLCL−T34	10TCLCL−T34	1
tCSH	9TCLCL−T26	9TCLCL−T26	11TCLCL−T26	11TCLCL−T26	1
tCAH	TCLCL−T34	2TCLCL−T34	2TCLCL−T34	2TCLCL−T34	1
tAR	2TCLCL−T26	3TCLCL−T26	3TCLCL−T26	3TCLCL−T26	1
tT	3/30	3/30	3/30	3/30	2
tRWC	12TCLCL	12TCLCL	14TCLCL	14TCLCL	1
tRRW	9TCLCL−T26	9TCLCL−T26	11TCLCL−T26	11TCLCL−T26	1
tCRW	8TCLCL−T34	8TCLCL−T34	10TCLCL−T34	10TCLCL−T34	1
tRCS	TCLCL−T36 −TBUF	TCLCL−T36 −TBUF	TCLCL−T36 −TBUF	TCLCL−T36 −TBUF	1
tRWD	6TCLCL−T26	6TCLCL−T26	8TCLCL−T26	8TCLCL−T26	1
tCWD	5TCLCL−T34	5TCLCL−T34	7TCLCL−T34	7TCLCL−T34	1
tWP	3TCLCL−T36 −TBUF	3TCLCL−T36 −TBUF	3TCLCL−T36 −TBUF	3TCLCL−T36 −TBUF	1
tRWL	3TCLCL−T36 −TBUF	3TCLCL−T36 −TBUF	3TCLCL−T36 −TBUF	3TCLCL−T36 −TBUF	1
tCWL	3TCLCL−T36 −TBUF	3TCLCL−T36 −TBUF	3TCLCL−T36 −TBUF	3TCLCL−T36 −TBUF	1

210463-005

Table 18 B. ECC Mode — RMW

Parameters	Slow Cycle Mode			
	C_4	C_5	C_6	Notes
tRP	2TCLCL−T26	2TCLCL−T26	2TCLCL−T26	1
tCPN	2.5TCLCL−T35	2.5TCLCL−T35	2.5TCLCL−T35	1
tRSH	7TCLCL−T34	7TCLCL−T34	5TCLCL−T34	1
tCSH	7TCLCL−T26	7TCLCL−T26	5TCLCL−T26	1
tCAH	2TCLCL−T34	2TCLCL−T34	2TCLCL−T34	1
tAR	2TCLCL−T26	2TCLCL−T26	2TCLCL−T26	1
tT	3/30	3/30	3/30	2
tRWC	9TCLCL	9TCLCL	7TCLCL	1
tRRW	7TCLCL−T26	7TCLCL−T26	5TCLCL−T26	1
tCRW	7TCLCL−T34	7TCLCL−T34	5TCLCL−T34	1
tRCS	0.5TCLCL−T36 −TBUF	0.5TCLCL−T36 −TBUF	0.5TCLCL−T36 −TBUF	1
tRWD	4TCLCL+TCL −T26	4TCLCL+TCL −T26	2TCLCL+TCL −T26	1
tCWD	4TCLCL+TCL −T34	4TCLCL+TCL −T34	2TCLCL+TCL −T34	1
tWP	3TCLCL−TCL −T36−TBUF	3TCLCL−TCL −T36−TBUF	3TCLCL−TCL −T36−TBUF	1
tRWL	3TCLCL−TCL −T36−TBUF	3TCLCL−TCL −T36−TBUF	3TCLCL−TCL −T36−TBUF	1
tCWL	3TCLCL−TCL −T36−TBUF	3TCLCL−TCL −T36−TBUF	3TCLCL−TCL −T36−TBUF	1

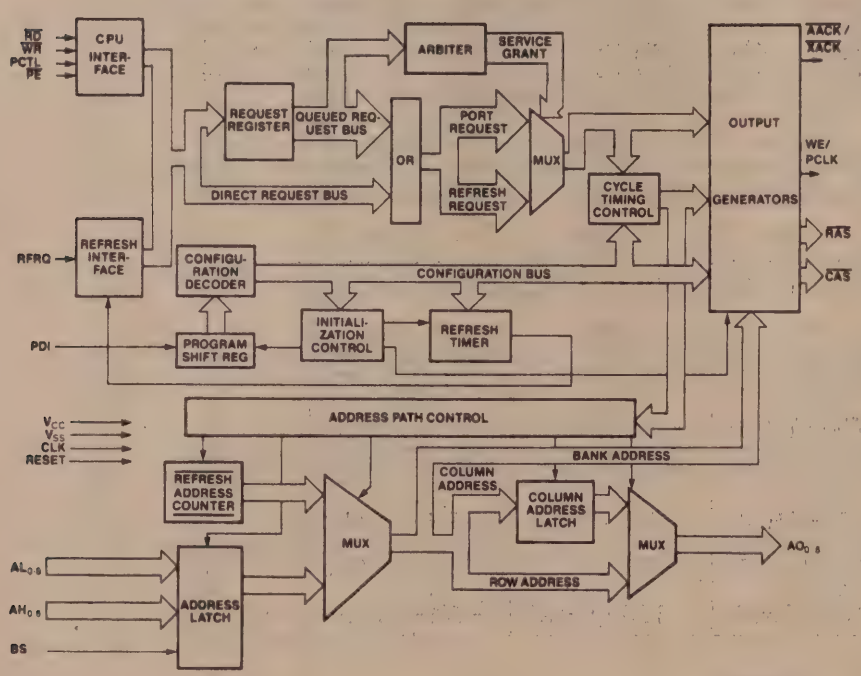
NOTES:
1. Minimum
2. Value on right is maximum; value on left is minimum.
3. Applies to the eight warm-up cycles during initialization only.
4. Applies to the eight warm-up cycles and to the memory initilization cycles during initialization only.
5. TP = TCLCL
   T26 = TCLRSL
   T34 = TCLCSL
   T35 = TCLCSH
   T36 = TCLW
   TBUF = TTL Buffer delay

210463-005

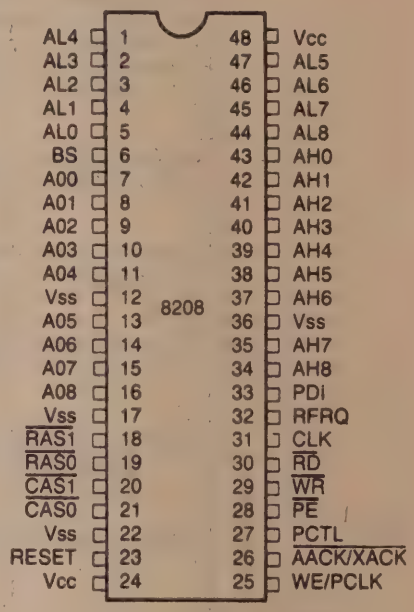
8208
DYNAMIC RAM CONTROLLER

- 0 Wait State, 8 Mhz iAPX 286, iAPX 186/188, and iAPX 86/88 Interface
- Provides all Signals necessary to Control 64k and 256k Dynamic RAMs
- Support Synchronous or Asynchronous Microprocessor Interfaces
- Automatic RAM Warm-up
- Performs Early Write Cycles
- iAPX 286 CFS = 1 (fast cycle)
  - 8208-16 4-16 MHz
  - 8208-12 4-12 MHz
- iAPX 86/186 CFS = 0 (slow cycle)
  - 8208 2-8 MHz
  - 8208-6 2-6 MHz
- Directly Addresses and Drives up to 1 Megabyte without External Drivers

The Intel 8208 Dynamic RAM Controller is a high performance, systems oriented, Dynamic RAM controller that is designed to easily interface 64k and 256k Dynamic RAMs to Intel and other microprocessors. It is a 48 pin single-port version of the dual-port 8207.



230734–1



230734–2

Figure 1. Block Diagram and Pinout Diagram

Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied. Information contained herein supersedes previously published specifications on these devices from Intel.
© Intel Corporation, 1985

March 1985
Order Number: 230734-003

Table 1. Pin Description

Symbol	Pin	Type	Name and Function
AL0 AL1 AL2 AL3 AL4 AL5 AL6 AL7 AL8	5 4 3 2 1 47 46 45 44	I I I I I I I I I	**ADDRESS LOW:** These low order address inputs are used to generate the row address for the internal address multiplexer. In iAPX 286 mode (CFS = 1), these addresses are latched internally.
BS	6	I	**BANK SELECT:** This input is used to select one of the two banks of the dynamic RAM array as defined by the program-bit RB.
AO0 AO1 AO2 AO3 AO4 AO5 AO6 AO7 AO8	7 8 9 10 11 13 14 15 16	O O O O O O O O O	**ADDRESS OUTPUTS:** These outputs are designed to provide the row and column addresses, of either the CPU or the refresh counter, to the dynamic RAM array. These outputs drive the dynamic RAM array directly and need no external drivers. However, they typically need series resistors to match impedances.
VSS	12 17 22 36	I I I I	**GROUND** **GROUND** **GROUND** **GROUND**
$\overline{RAS0}$ $\overline{RAS1}$	19 18	O O	**ROW ADDRESS STROBE:** These outputs are used by the dynamic RAM array to latch the row address, present on the AO0–8 pins. These outputs are selected by the BS pin as programmed by program-bit RB. These outputs drive the dynamic RAM array directly and need no external drivers.
$\overline{CAS0}$ $\overline{CAS1}$	21 20	O O	**COLUMN ADDRESS STROBE:** These outputs are used by the dynamic RAM array to latch the column address, present on the AO0–8 pins. These outputs are selected by the BS pin as programmed by program-bit RB. These outputs drive the dynamic RAM array directly and need no external drivers.
RESET	23	I	**RESET:** This active high signal causes all internal counters to be reset and upon release of RESET, data appearing at the PDI pin is clocked-in by the PCLK output. The states of the PDI, PCTL and RFRQ pins are sampled by RESET going inactive and are used to program the 8208. An 8 cycle dynamic RAM warm-up is performed after clocking PDI bits into the 8208.
WE/ PCLK	25	O	**WRITE ENABLE/PROGRAMMING CLOCK:** Immediately after a RESET this pin becomes PCLK and is used to clock serial programming data into the PDI pin. After the 8208 is programmed this active high signal provides the dynamic RAM array the write enable input for a write operation.
VCC	24 48	I I	**POWER:** +5 Volts. **POWER:** +5 Volts.

Table 1. Pin Description (Continued)

Symbol	Pin	Type	Name and Function
$\overline{AACK}$/ $\overline{XACK}$	26	O	**ADVANCE ACKNOWLEDGE/TRANSFER ACKNOWLEDGE:** When the X programming bit is set to logic 0 this pin is $\overline{AACK}$ and indicates that the processor may continue processing and that data will be available when required. This signal is optimized for the system by programming the S program-bit for synchronous or asynchronous operation. The S programming bit determines whether this strobe will be early or late. If another dynamic RAM cycle is in progress at the time of the new request, the $\overline{AACK}$ is delayed. When the X programming bit is set to logic 1 this pin is $\overline{XACK}$ and indicates that data on the bus is valid during a read cycle or that data may be removed from the bus during a write cycle. $\overline{XACK}$ is a MULTIBUS compatible signal. (See Figure 5)
PCTL	27	I	**PORT CONTROL:** This pin is sampled on the falling edge of RESET. It configures the 8208 to accept command inputs or processor status inputs. If PCTL is low after RESET the 8208 is programmed to accept bus/MULTIBUS command inputs or iAPX 286 status inputs. If PCTL is high after RESET the 8208 is programmed to accept status inputs from iAPX 86 or iAPX 186 type processors. The $\overline{S2}$ status line should be connected to this input if programmed to accept iAPX 86 or iAPX 186 status inputs. When programmed to accept bus commands or iAPX 286 status inputs, it should be tied low or it may be connected to INHIBIT when operating with MULTIBUS.
$\overline{PE}$	28	I	**PORT ENABLE:** This pin serves to enable a RAM cycle request. It is generally decoded from the address bus.
$\overline{WR}$	29	I	**WRITE:** This pin is the write memory request command input. This input also directly accepts the $\overline{S0}$ status line from Intel processors.
$\overline{RD}$	30	I	**READ:** This pin is the read memory request command input. This input also directly accepts the $\overline{S1}$ status line from Intel processors.
CLK	31	I	**CLOCK:** This input provides the basic timing for sequencing the internal logic. This clock requires MOS levels.
RFRQ	32	I	**REFRESH REQUEST:** This input is sampled on the falling edge of RESET. If RFRQ is high at RESET then the 8208 is programmed for internal-refresh request or external-refresh request with failsafe protection. If RFRQ is low at RESET then the 8208 is programmed for external-refresh without failsafe protection or burst-refresh. Once programmed the RFRQ pin accepts signals to start an external-refresh with failsafe protection or external-refresh without failsafe protection or a burst-refresh.
PDI	33	I	**PROGRAM DATA INPUT:** This input is sampled by RESET going low. It programs the various user selectable options in the 8208. The PCLK pin shifts programming data into the PDI input from an external shift register. This pin may be strapped low to a default iAPX 186 mode configuration or high to a default iAPX 286 mode configuration.
AH0 AH1 AH2 AH3 AH4 AH5 AH6 AH7 AH8	43 42 41 40 39 38 37 35 34	I I I I I I I I I	**ADDRESS HIGH:** These higher order address inputs are used to generate the column address for the internal address multiplexer. In iAPX 286 mode, these addresses are latched internally.

GENERAL DESCRIPTION

The Intel 8208 Dynamic RAM Controller is a microcomputer peripheral device which provides the necessary signals to address, refresh and directly drive 64k and 256k dynamic RAMs.

The 8208 supports several microprocessor interface options including synchronous and asynchronous operations for iAPX 286, iAPX 186/188, iAPX 86/88 and MULTIBUS.

FUNCTIONAL DESCRIPTION

Processor Interface

The 8208 has control circuitry capable of supporting one of several possible bus structures. The 8208 may be programmed to run synchronous or asynchronous to the processor clock. (See Synchronous/Asynchronous Mode.) The 8208 has been optimized to run synchronously with Intel's iAPX 286, iAPX 186/188, and iAPX 86/88. When the 8208 is programmed to run in asynchronous mode, the 8208 inserts the necessary synchronization circuitry for the $\overline{RD}$, $\overline{WR}$, $\overline{PE}$, and PCTL inputs.

The 8208 achieves high performance (i.e. no wait states) by decoding the status lines directly from the iAPX 286, iAPX 186/188, and the iAPX 86/88. The 8208 can also be programmed to receive read or write MULTIBUS commands or commands from a bus controller. (See Status/Command Mode.)

The 8208 may be programmed to accept the clock of any Intel microprocessor. The 8208 adjusts its internal timing to allow for different clock frequencies of these microprocessors. (See Microprocessor Clock Frequency Option.)

Figure 2 shows the different processor interfaces to the 8208 using the synchronous or asynchronous mode and status or command interface.

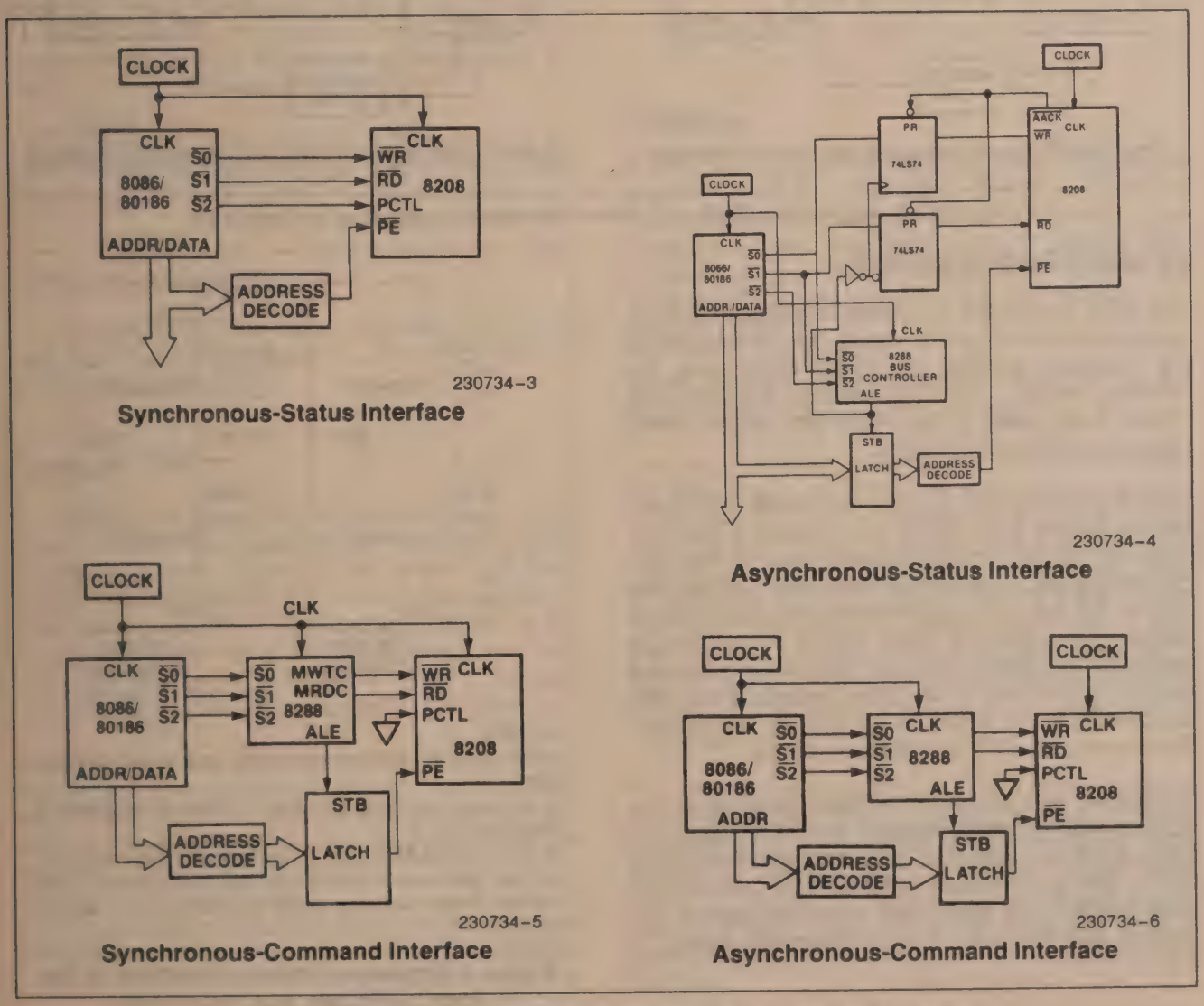


Figure 2A. Slow-Cycle (CFS = 0) Interfaces Supported by the 8208

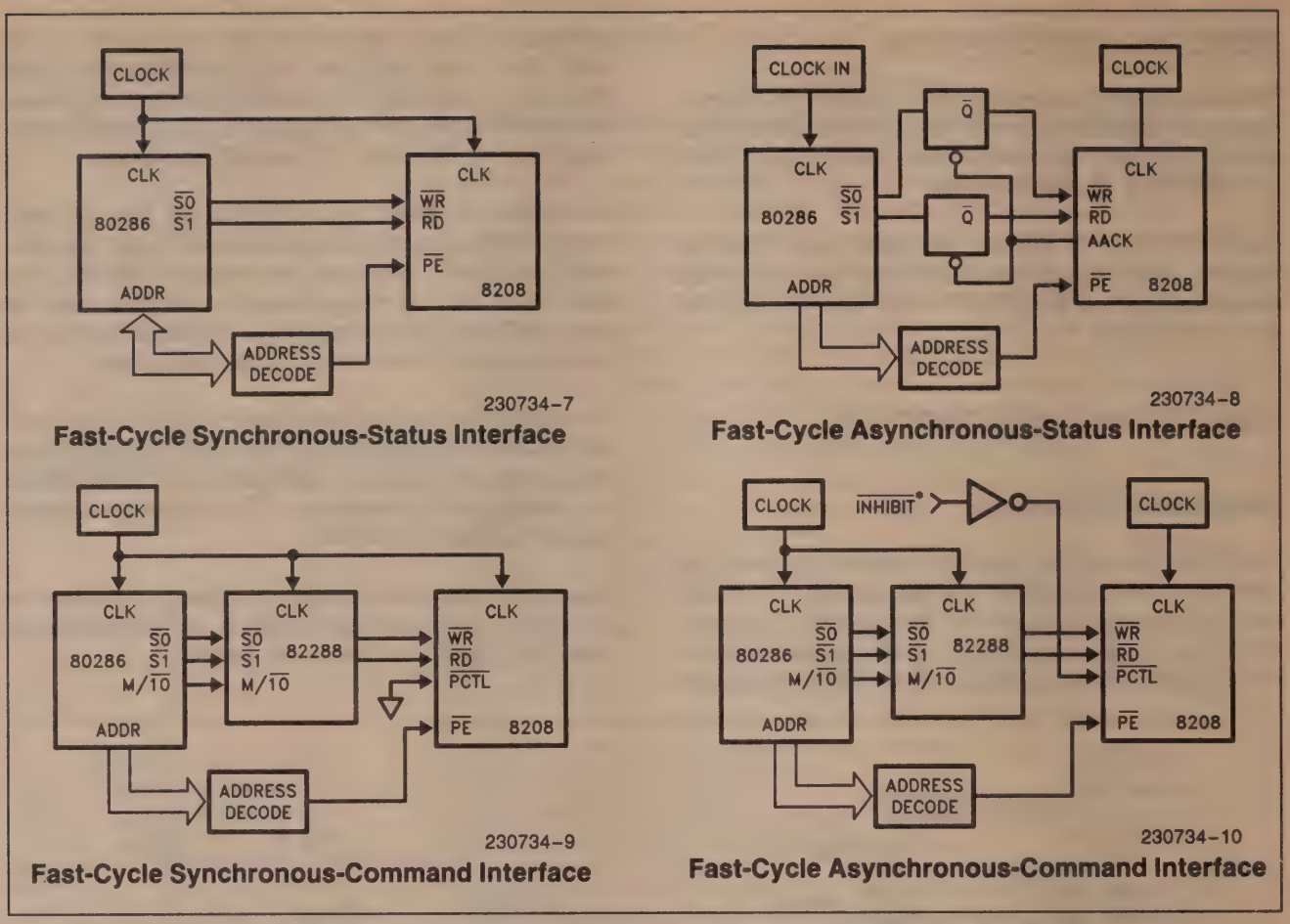


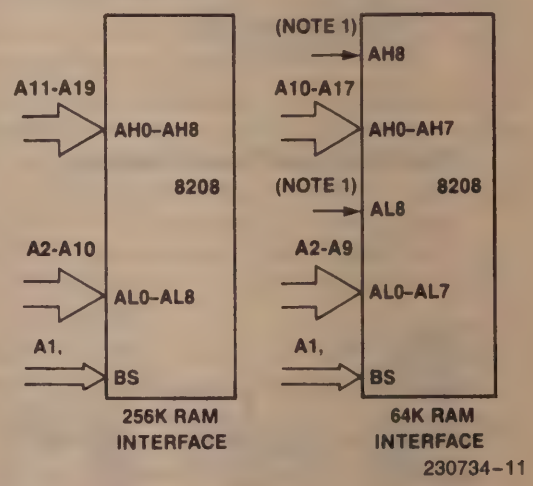
Figure 2B. Fast-cycle (CFS = 1) Port Interfaces Supported by the 8208

Dynamic RAM Interface

The 8208 is capable of addressing 64k and 256k dynamic RAMs. Figure 3 shows the connection of the processor address bus to the 8208 using the different RAMs. The 8208 directly supports the 2164A RAM family or any RAM with similar timing requirements and responses.

The 8208 divides memory into two banks, each bank having its own Row ($\overline{RAS}$) and Column ($\overline{CAS}$) Address Strobe pair. This organization permits RAM cycle interleaving. RAM cycle interleaving overlaps the start of the next RAM cycle with the RAM precharge period of the previous cycle. Hiding the precharge period of one RAM cycle behind the data access period of the next RAM cycle optimizes memory bandwidth and is effective as long as successive RAM cycles occur in the alternate banks.

Successive data access to the same bank cause the 8208 to wait for the precharge time of the previous RAM cycle. But when the 8208 is programmed in an iAPX 186 synchronous configuration consecutive read cycles to the same bank does not result in additional wait states (i.e. 0 wait state reads result).



NOTES:
1. Unassigned address input pins should be strapped high or low.
2. A0 along with BHE are used to select a byte within a processor word.
3. Low order address bit is used as a bank select input so that consecutive memory access requests are to alternate banks allowing bank interleaving of memory cycles.

Figure 3. Processor Address Interface to the 8208 Using 64k, and 256k, RAMS

If not all RAM banks are occupied, the 8208 reassigns the $\overline{RAS}$ and $\overline{CAS}$ strobes to allow using wider data words without increasing the loading on the $\overline{RAS}$ and $\overline{CAS}$ drivers. Table 2 shows the bank selection decoding and the horizontal word expansion, including $\overline{RAS}$ and $\overline{CAS}$ assignments. For example, if only one RAM bank is occupied, then the two $\overline{RAS}$ and $\overline{CAS}$ strobes are activated with the same timing.

Table 2. Bank Selection Decoding and Word Expansion

Program Bit RB	Bank Input BS	8208 $\overline{RAS}/\overline{CAS}$ Pair Allocation
0	0	$\overline{RAS}_{0,1}$, $\overline{CAS}_{0,1}$ to Bank 0
0	1	Illegal
1	0	$\overline{RAS}_0$, $\overline{CAS}_0$ to Bank 0
1	1	$\overline{RAS}_1$, $\overline{CAS}_1$ to Bank 1

Program bit RB is not used to check the bank select input BS. The system design must protect from accesses to "illegal", non-existent banks of memory by deactivating the PE input when addressing an "illegal", non-existent bank of memory.

The 8208 adjusts and optimizes internal timings for either the fast or slow RAMs as programmed. (See RAM Speed Option.)

Memory Initialization

After programming, the 8208 performs eight RAM "wake-up" cycles to prepare the dynamic RAM for proper device operation.

Refresh

The 8208 provides an internal refresh interval counter and a refresh address counter to allow the 8208 to refresh memory. The 8208 will refresh 128 rows every 2 milliseconds or 256 rows every 4 milliseconds, which allows all RAM refresh options to be supported. In addition, there exists the ability to refresh 256 row address locations every 2 milliseconds via the Refresh Period programming option.

The 8208 may be programmed for any of five different refresh options: Internal refresh only, External refresh with failsafe protection, External refresh without failsafe protection, Burst Refresh modes, or no refresh. (See Refresh Options.)

It is possible to decrease the refresh time interval by 10%, 20% or 30%. This option allows the 8208 to compensate for reduced clock frequencies. Note that an additional 5% interval shortening is built-in in all refresh interval options to compensate for clock variations and non-immediate response to the internally generated refresh request. (See Refresh Period Options.)

External Refresh Requests after RESET

External refresh requests are not recognized by the 8208 until after it is finished programming and preparing memory for access. Memory preparation includes 8 RAM cycles to prepare and ensure proper dynamic RAM operation. The time it takes for the 8208 to recognize a request is shown below.

eq. 8208 System Response:

$$TRESP = TPROG + TPREP$$

where: TPROG = (40) (TCLCL) which is programming time

TPREP = (8) (32) (TCLCL) which is the RAM warm-up time

if TCLCL = 125 ns then TRESP = 37 us

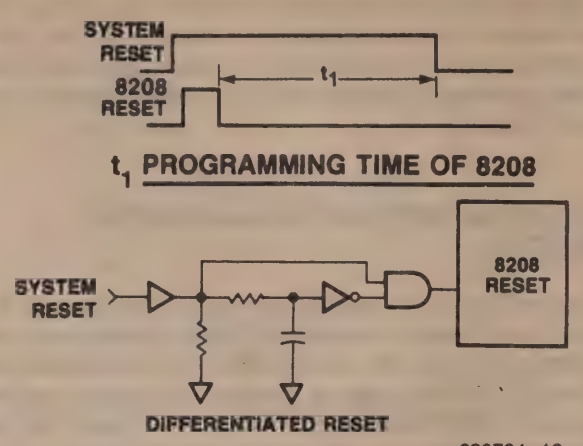
Reset

RESET is an asynchronous input, the falling edge of which is used by the 8208 to directly sample the logic levels of the PCTL, RFRQ, and PDI inputs. The internally synchronized falling edge of reset is used to begin programming operations (shifting in the contents of the external shift register, if needed, into the PDI input).

Differentiated reset is unnecessary when the default synchronization programming is used.

Until programming is complete the 8208 registers but does not respond to command or status inputs. A simple means of preventing commands or status from occurring during this period is to differentiate the system reset pulse to obtain a smaller reset pulse for the 8208. The total time of the 8208 reset pulse and the 8208 programming time must be less than the time before the first command the CPU issues in systems that alter the default port synchronization programming bit (default is synchronous interface).

The differentiated reset pulse would be shorter than the system reset pulse by at least the programming period required by the 8208. The differentiated reset pulse first resets the 8208, and system reset would reset the rest of the system. While the rest of the system is still in reset, the 8208 completes its programming. Figure 4 illustrates a circuit to accomplish this task.



NOTES:
1. Required only when the synchronization option is altered from its initial default value.
2. V_{CC} must be stable before system reset is activated when using this circuit.

Figure 4. 8208 Differentiated Reset Circuit

Within four clocks after RESET goes active, all the 8208 outputs will go high, except for AO0–2, which will go low.

OPERATIONAL DESCRIPTION

Programming the 8208

The 8208 is programmed after reset. On the falling edge of RESET, the logic states of several input pins are latched internally. The falling edge of RESET actually performs the latching, which means that the logic levels on these inputs must be stable prior to that time. The inputs whose logic levels are latched at the end of reset are the PCTL, REFRQ, and PDI pins.

Status/Command Mode

The processor port of the 8208 is configured by the states of the PCTL pin. Which interface is selected depends on the state of the PCTL pin at the end of reset. If PCTL is high at the end of reset, the 8086/80186 Status interface is selected; if it is low, then the MULTIBUS or Command interface is selected.

The status lines of the 80286 are similar in code and timing to the Multibus command lines, while the status code and timing of the 8086 and 8088 are identical to those of the 80186 and 80188 (ignoring the differences in clock duty cycle). Thus there exists two interface configurations, one for the 80286 status or Multibus memory commands, which is called the Command interface, and one for 8086, 8088, 80186 or 80188 status, called the 8086 Status interface. The Command interface can also directly interface to the command lines of the bus controllers for the 8086, 8088, 80186 and the 80286.

The 80186 Status interface allows direct decoding of the status lines for the iAPX 86, iAPX 88, iAPX 186 and the iAPX 188. Table 3 shows how the status lines are decoded. Microprocessor bus controller read or write commands or MULTIBUS commands can also be directed to the 8208 when in Command mode.

Table 3A. Status Coding of 8086, 80186 and 80286

Status Code			Function	
$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	8086/80186	80286*
0	0	0	INTERRUPT	INTERRUPT
0	0	1	I/O READ	I/O READ
0	1	0	I/O WRITE	I/O WRITE
0	1	1	HALT	IDLE
1	0	0	INSTRUCTION FETCH	HALT
1	0	1	MEMORY READ	MEMORY READ
1	1	0	MEMORY WRITE	MEMORY WRITE
1	1	1	IDLE	IDLE

* Refer 80286 pin description table

Table 3B. 8208 Response

8208 Command			Function	
PCTL	$\overline{RD}$	$\overline{WR}$	8086/80186 Status Interface	80286 Status or Command Interface
0	0	0	IGNORE	IGNORE*
0	0	1	IGNORE	READ
0	1	0	IGNORE	WRITE
0	1	1	IGNORE	IGNORE
1	0	0	READ	IGNORE
1	0	1	READ	INHIBIT
1	1	0	WRITE	INHIBIT
1	1	1	IGNORE	IGNORE

*Illegal with CFS = 0

Refresh Options

Immediately after system reset, the state of the REFRQ input pin is examined. If REFRQ is high, the 8208 provides the user with the choice between self-refresh and user-generated refresh with failsafe protection. Failsafe protection guarantees that if the user does not come back with another refresh

request before the internal refresh interval counter times out, a refresh request will be automatically generated. If the REFRQ pin is low immediately after a reset, then the user has the choice of a single external refresh cycle without failsafe, burst refresh or no refresh.

Internal Refresh Only

For the 8208 to generate internal refresh requests, it is necessary only to strap the REFRQ input pin high.

External Refresh with Failsafe

To allow user-generated refresh requests with failsafe protection, it is necessary to hold the REFRQ input high until after reset. Thereafter, a low-to-high transition on this input causes a refresh request to be generated and the internal refresh interval counter to be reset. A high-to-low transition has no effect on the 8208. A refresh request is not recognized until a previous request has been serviced.

External Refresh without Failsafe

To generate single external refresh requests without failsafe protection, it is necessary to hold REFRQ low until after reset. Thereafter, bringing REFRQ high for one clock period will cause a refresh request to be generated. A refresh request is not recognized until a previous request has been serviced.

Burst Refresh

Burst refresh is implemented through the same procedure as a single external refresh without failsafe (i.e., REFRQ is kept low until after reset). Thereafter, bringing REFRQ high for at least two clock periods will cause a burst of up to 128 row address locations to be refreshed. Any refresh request is not recognized until a previous request has been serviced (i.e. burst is completed).

No Refresh

It is necessary to hold REFRQ low until after reset. This is the same as programming External Refresh without Failsafe. No refresh is accomplished by keeping REFRQ low.

Option Program Data Word

The program data word consists of 9 program data bits, PD0–PD8. If the first program data bit, PD0 is set to logic 0, the 8208 is configured to support iAPX 186, 188, 86, or 88 systems. The remaining bits, PD1–PD8, may then be programmed to optimize a selected system configuration. A default of all zeros in the remaining program bits optimizes the 8208 timing for 8 MHz Intel CPUs using 150 ns (or faster) dynamic RAMs with no performance penalty.

If the first program data bit is set to logic 1, the 8208 is configured to support iAPX 286 systems (Command mode). A default of all ones in the program bits optimizes the 8208 timing for an 8 MHz 286 using 120 ns DRAMs at zero wait states. Note that the programming bits PD1–8 change polarity according to PD0. This ensures the same choice of options for both default modes.

Figure 5 shows the various options that can be programmed into the 8208.

Figure 5. Program Data Word

Program Data Bit	Name		Polarity/Function
	PD0 = 0	PD0 = 1	
PD0	CFS	CFS	CFS = 0 SLOW CYCLE CFS = 1 FAST CYCLE
PD1	$\overline{S}$	S	$\overline{S}$ = 0 SYNCHRONOUS* $\overline{S}$ = 1 ASYNCHRONOUS
PD2	$\overline{RFS}$	RFS	$\overline{RFS}$ = 0 FAST RAM* $\overline{RFS}$ = 1 SLOW RAM
PD3	$\overline{RB}$	RB	RAM BANK OCCUPANCY SEE TABLE 2
PD4	CI1	$\overline{CI1}$	COUNT INTERVAL BIT 1; SEE TABLE 6
PD5	CI0	$\overline{CI0}$	COUNT INTERVAL BIT 0; SEE TABLE 6
PD6	$\overline{PLS}$	PLS	$\overline{PLS}$ = 0 LONG REFRESH PERIOD* $\overline{PLS}$ = 1 SHORT REFRESH PERIOD
PD7	$\overline{FFS}$	FFS	$\overline{FFS}$ = 0 FAST CPU FREQUENCY* $\overline{FFS}$ = 1 SLOW CPU FREQUENCY
PD8	X	$\overline{X}$	X = 0 $\overline{AACK}$* X = 1 $\overline{XACK}$

* Default in both modes

Using an External Shift Register

The 8208 may be programmed by using an external shift register with asynchronous load capability such as a 74LS165. The reset pulse serves to parallel load the shift register and the 8208 supplies the clocking signal (PCLK) to shift the data into the PDI

programming pin. Figure 6 shows a sample circuit diagram of an external shift register circuit.

Serial data is shifted into the 8208 via the PDI pin (33), and clock is provided by the WE/PCLK pin (25), which generates a total of 9 clock pulses. After programming is complete, data appearing at the input of the PDI pin is ignored.

WE/PCLK is a dual function pin. During programming, it serves to clock the external shift register, and after programming is completed, it reverts to the write enable RAM control output pin. As the pin changes state to provide the write enable signal to the dynamic RAM array, it continues to clock the shift register. This does not present a problem because data at the PDI pin is ignored after programming. Figure 7 illustrates the timing requirements of the shift register.

Default Programming Options

After reset, the 8208 serially shifts in a program data word via the PDI pin. This pin may be strapped low or high, or connected to an external shift register. Strapping PDI low causes the 8208 to default to the iAPX 186 system configuration, while high causes a

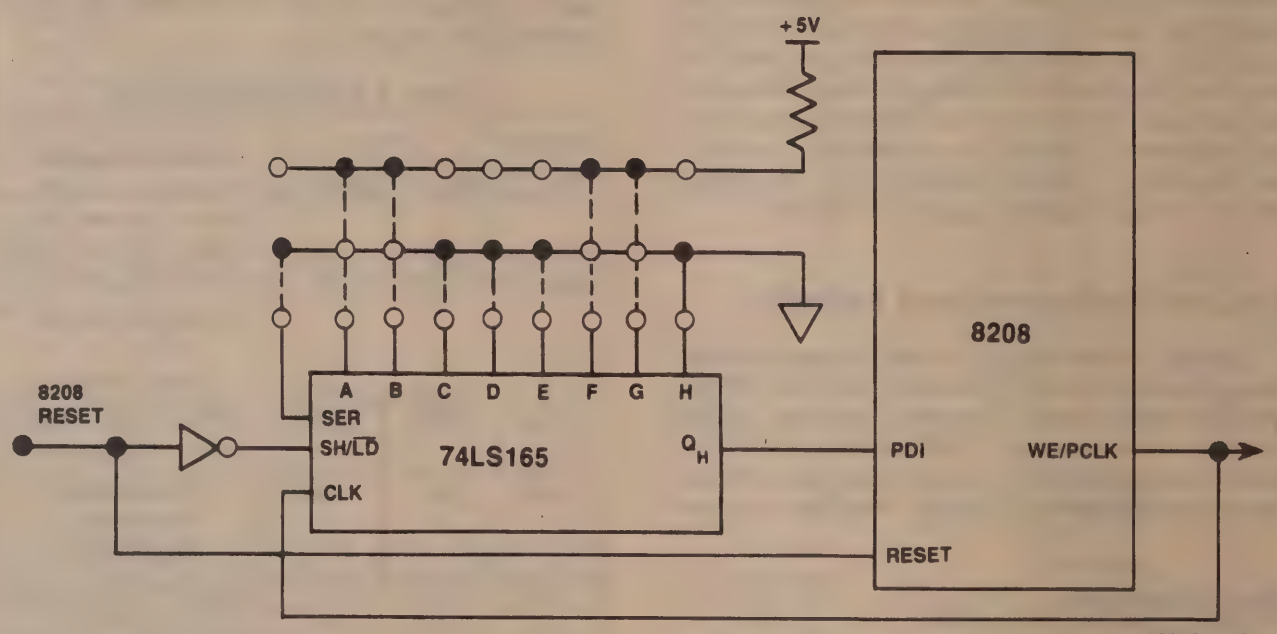
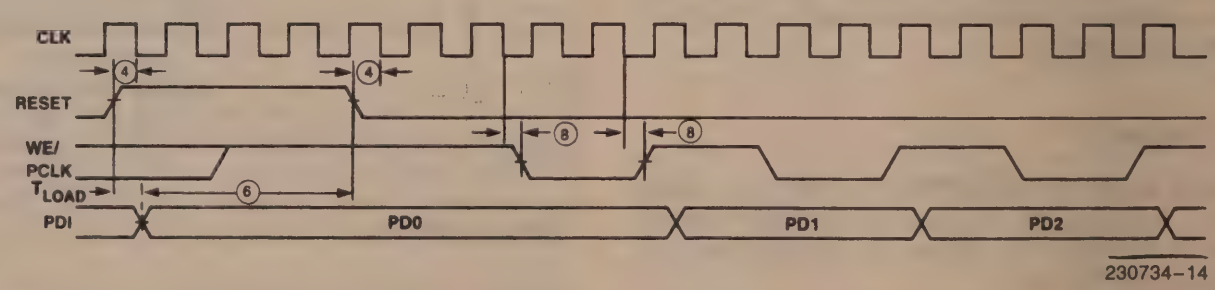


Figure 6. External Shift Register Interface



NOTES:
TRTVCL — Reset is an asychronous input, if reset occurs before TRTVCL, then it is guaranteed to be recognized.
TPGVCL — Minimum PDI valid time prior to reset going low.
TCLPC — MUX/PCLK delay.
TLOAD — Asychronous load data propagation delay.

Figure 7. Timing Illustrating External Shift Register Requirements for Programming the 8208

default to the iAPX 286 configuration. Table 4 shows the characteristics of the default configuration. If further system flexibility is needed, one external shift register, like a 74LS165, can be used to tailor the 8208 to its operating environment.

Table 4. Default Programming

Synchronous interface
Fast RAM (Note 1)
2 RAM banks occupied
Refresh interval uses 118 clocks
128 row refresh in 2 ms; 256 row refresh in 4 ms
Fast processor clock frequency (8 MHz)
Advanced ACK strobe

NOTE:
1. For iAPX 86/186 systems either slow or fast (150 or 100 ns) RAMS will run at 8 MHz with zero wait states.

Synchronous/Asynchronous Mode (S program bit)

The 8208 may be independently configured to accept synchronous or asynchronous commands ($\overline{RD}$, $\overline{WR}$, PCTL) and Port Enable ($\overline{PE}$) via the S program bit. The state of the S programming bit determines whether the interface is synchronous or asynchronous.

While the 8208 may be configured with either the Status or Command (MULTIBUS) interface in the Synchronous mode, certain restrictions exist in the Asynchronous mode. An Asynchronous-Command interface using the control lines of the MULTIBUS is supported, and an Asynchronous-80186 Status interface using the status lines of the 80186 is supported, with the use of TTL gates as illustrated in Figure 2. In the 80186 case, the TTL gates are needed to guarantee that status does not appear at the 8208's inputs too much before address, so that a cycle would start before address was valid.

Microprocessor Clock Cycle Option (CFS and FFS program bits)

The 8208 is programmed to interface with microprocessors with "slow cycle" timing like the 8086, 8088, 80186, and 80188, and with "fast cycle" microprocessors like the 286. The CFS bit is used to select the appropriate timing.

The FFS option is used to select the speed of the microprocessor clock. Table 5 shows the various microprocessor clock frequency options that can be programmed. The external clock frequency must be programmed so that the failsafe refresh repetition circuitry can adjust its internal timing accordingly to produce a refresh request as programmed.

Table 5. Microprocessor Clock Frequency Options

Program Bits		Processor	Clock Frequency
CFS	FFS		
0	0	iAPX 86, 88, 186, 188	5 MHz
0	1	iAPX 86, 88, 186, 188	8 MHz
1	0	iAPX 286	10 MHz
1	1	iAPX 286	16 MHz

RAM Speed Option (RFS program bit)

The RAM Speed programming option determines whether RAM timing will be optimized for a fast or slow RAM. Whether a RAM is fast or slow is measured relative to 100 ns DRAMs (fast) or 150 ns DRAMs (slow). This option is only a factor in Command Mode (CFS = 1).

Refresh Period Options (CI0, CI1 and PLS program bits)

The 8208 refreshes with either 128 rows every 2 milliseconds or with 256 rows every 4 milliseconds. This translates to one refresh cycle being executed approximately once every 15.6 microseconds. This rate can be changed to 256 rows every 2 milliseconds or a refresh approximately once every 7.8 microseconds via the Period Long/Short, program bit PLS, programming option.

The Count Interval 0 (CI0) and Count Interval 1 (CI1) programming options allow the rate at which refresh requests are generated to be increased in order to permit refresh requests to be generated close to the 15.6 or 7.8 microsecond period when the 8208 is operating at reduced frequencies. The interval between refreshes is decreased by 0%, 10%, 20%, or 30% as a function of how the count interval bits are programmed. A 5% guardband is built-in to allow for any clock frequency variations. Table 6 shows the refresh period options available.

The numbers tabulated under Count Interval represent the number of clock periods between internal refresh requests. The percentages in parentheses represent the decrease in the interval between refresh requests. Note that all intervals have a built-in 5% (approximately) safety factor to compensate for minor clock frequency deviations and non-immediate response to internal refresh requests.

Table 6. Refresh Count Interval Table

Ref. Period (μS)	CFS	PLS	FFS	Count Interval CI1, CI0 (8208 Clock Periods)			
				00 (0%)	01 (10%)	10 (20%)	11 (30%)
15.6	1	1	1	236	212	188	164
7.8	1	0	1	118	106	94	82
15.6	1	1	0	148	132	116	100
7.8	1	0	0	74	66	58	50
15.6	0	1	1	118	106	94	82
7.8	0	0	1	59	53	47	41
15.6	0	1	0	74	66	58	50
7.8	0	0	0	37	33	29	25

Processor Timing

In order to run without wait states, $\overline{AACK}$ must be used and connected to the $\overline{SRDY}$ input of the appropriate bus controller. $\overline{AACK}$ is issued relative to a point within the RAM cycle and has no fixed relationship to the processors's request. The timing is such, however, that the processor will run without wait states, barring refresh cycles. In slow cycle, fast RAM configurations (8086, 80186), $\overline{AACK}$ is issued on the same clock cycle that issues $\overline{RAS}$.

Port Enable ($\overline{PE}$) set-up time requirements depend on whether the 8208 is configured for synchronous or asynchronous, fast or slow cycle operation. In a synchronous fast cycle configuration, $\overline{PE}$ is required to be set-up to the same clock edge as the commands. If $\overline{PE}$ is true (low), a RAM cycle is started; if not, the cycle is not started until the $\overline{RD}$ or $\overline{WR}$ line goes inactive.

In asychronous operation, $\overline{PE}$ is required to be set-up to the same clock edge as the internally synchronized status or commands. Externally, this allows the internal synchronization delay to be added to the status (or command) -to-$\overline{PE}$ delay time, thus allowing for more external decode time than is available in synchronous operation.

The minimum synchronization delay is the additional amount that $\overline{PE}$ must be held valid. If $\overline{PE}$ is not held valid for the maximum synchronization delay time, it is possible that $\overline{PE}$ will go invalid prior to the status or command being synchronized. In such a case the 8208 does not start a memory cycle. If a memory cycle intended for the 8208 is not started, then no acknowledge ($\overline{AACK}$ or $\overline{XACK}$) is issued and the processor locks up in endless wait states.

Memory Acknowledge ($\overline{AACK}$, $\overline{XACK}$)

Two types of memory acknowledge signals are supplied by the 8208. They are the Advanced Acknowledge strobe ($\overline{AACK}$) and the Transfer Acknowledge strobe ($\overline{XACK}$). The S programming bit optimizes $\overline{AACK}$ for synchronous operation ("early" $\overline{AACK}$) or asynchronous operation ("late" $\overline{AACK}$). Both the early and late $\overline{AACK}$ strobes are two clocks long for CFS = 0 and three clocks long for CFS = 1.

The $\overline{XACK}$ strobe is asserted when data is valid (for reads) or when data may be removed (for writes) and meets the MULTIBUS requirements. $\overline{XACK}$ is removed asynchronously by the command going inactive.

Since in an asynchronous operation the 8208 removes read data before late $\overline{AACK}$ or $\overline{XACK}$ is recognized by the CPU, the user must provide for data latching in the system until the CPU reads the data. In synchronous operation data latching is unnecessary, since the 8208 will not remove data until the CPU has read it.

If the X programming bit is high, the strobe is configured as $\overline{XACK}$, while if the bit is low, the strobe is configured as $\overline{AACK}$.

Data will always be valid a fixed time after the occurrence of the advanced acknowledge. Thus, the advanced acknowledge may also serve as a RAM cycle timing indicator.

Table 7. Memory Acknowledge Summary

	Synchronous	Asynchronous	XACK
Fast Cycle	AACK Optimized for Local 80286 (early)	AACK Optimized for Remote 80286 (late)	Multibus Compatible
Slow Cycle	AACK Optimized for Local 8086/186 (early)	AACK Optimized for Remote 8086/186 (late)	Multibus Compatible

General System Considerations

1. The $\overline{RAS}$0, 1, $\overline{CAS}$0, 1 and AO0–8 output buffers are designed to directly drive the heavy capacitive loads associated with dynamic RAM arrays. To keep the RAM driver outputs from ringing excessively in the system environment it is necessary to match the output impedance with the RAM array by using series resistors. Each application may have different impedance characteristics and may require different series resistance values. The series resistance values should be determined for each application.
2. Although the 8208 has programmable options, in practice there are only a few choices the designer must make. For iAPX 86/186 systems (CFS = 0), the C2 default mode (pin 33 tied low) is the best choice. This permits zero wait states at 8 MHz with 150 ns DRAM's. The only consideration is the refresh rate, which must be programmed if the CPU is run at less than 8 MHz.

For iAPX 286 systems (CFS = 1) the designer must choose between configuration C0 ($\overline{RFS}$ = 0) and C1 ($\overline{RFS}$ = 1, $\overline{FFS}$ = 0). C0 permits zero wait state, 8 MHz iAPX 286 operaton with 120 ns DRAM's. However, for consecutive reads, this performance depends upon interleaving between two banks. The C1 configuration trades off 1 wait state performance for the ability to use 150 ns DRAM's. 150 ns DRAMs can be supported by the C0 configuration using 7 MHz iAPX 286. Finally, for non-Intel processors the usual choice is asynchronous, command mode (C0), since status lines are not available, Typically, to minimize the 8208's synchronization delay, the 8208 would be run as fast as possible.

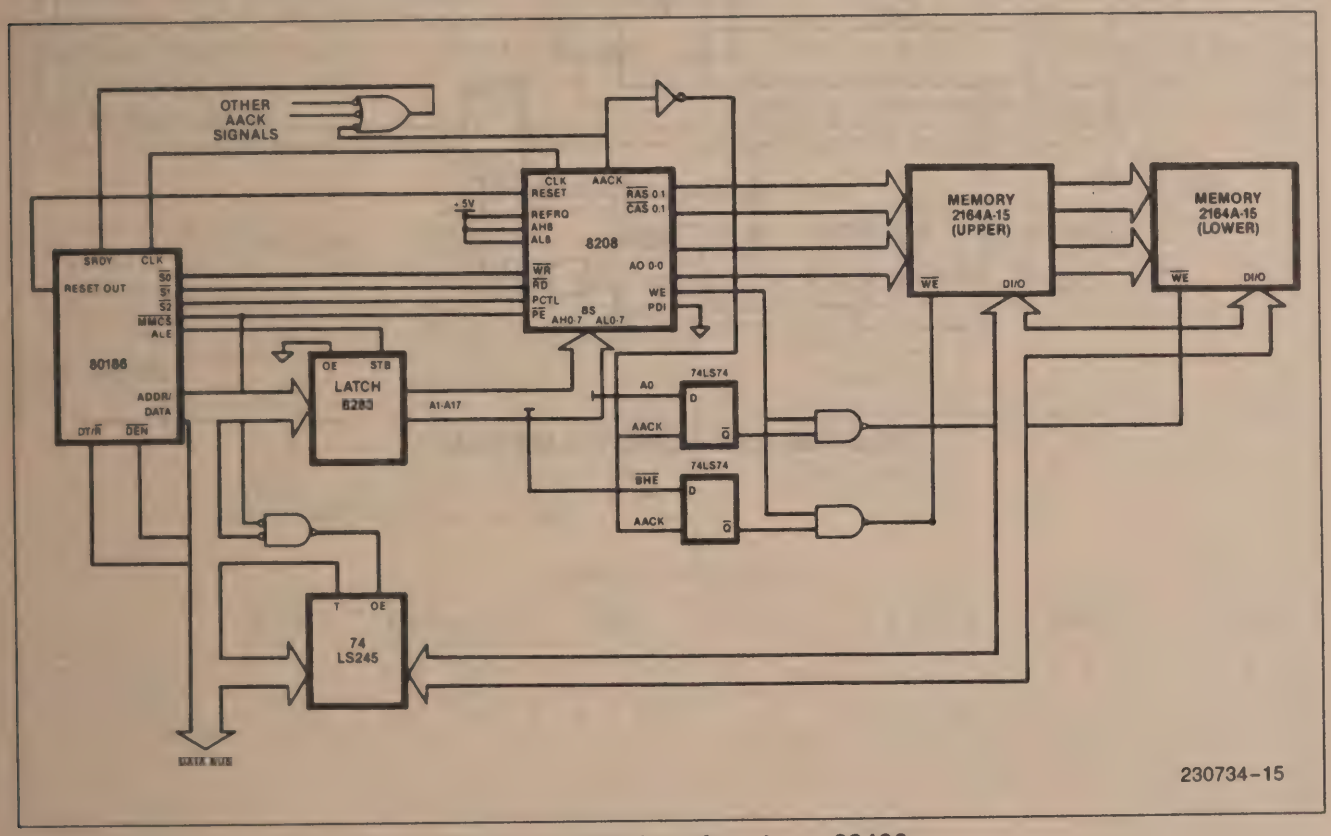


Figure 8A. 8208 Interface to an 80186

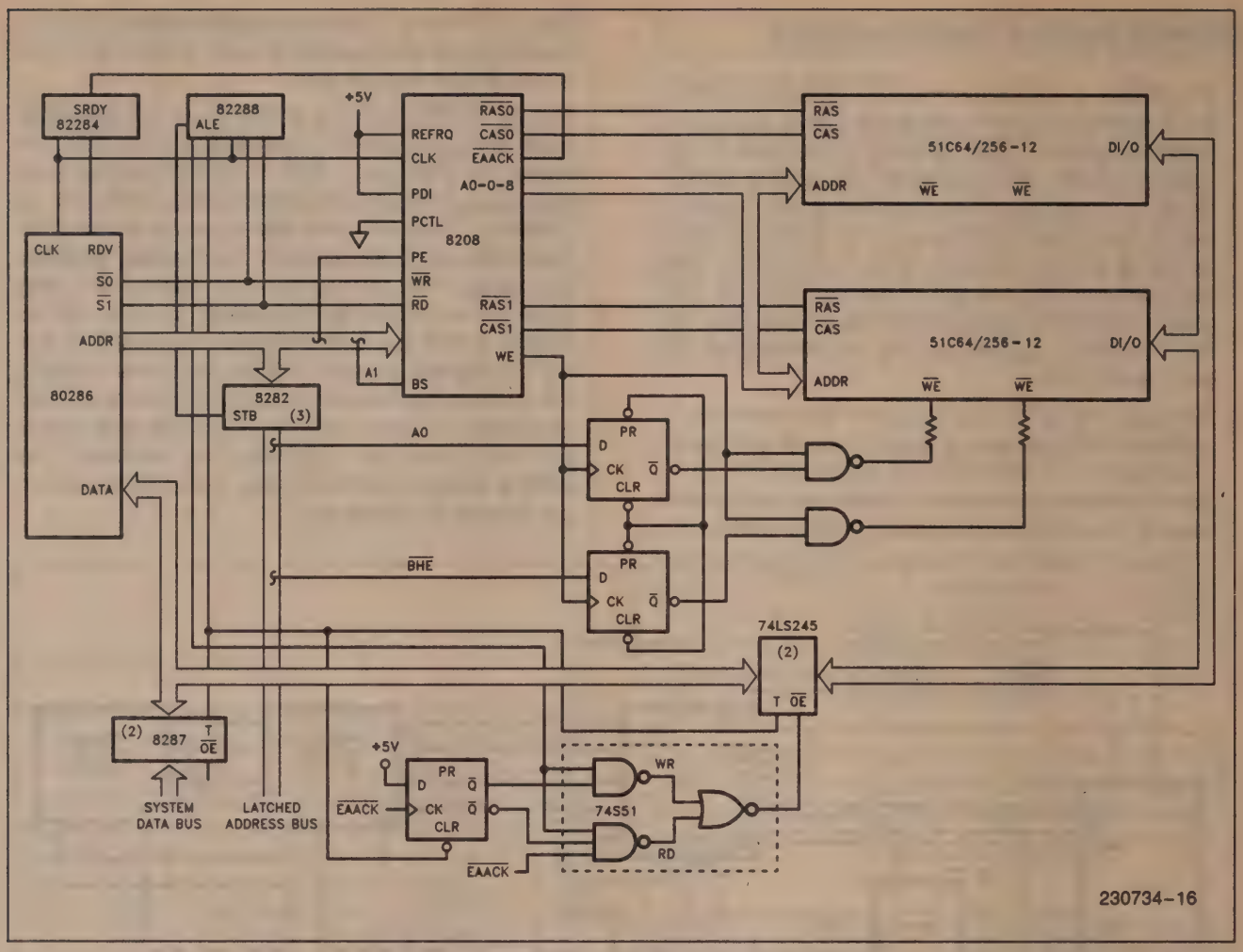


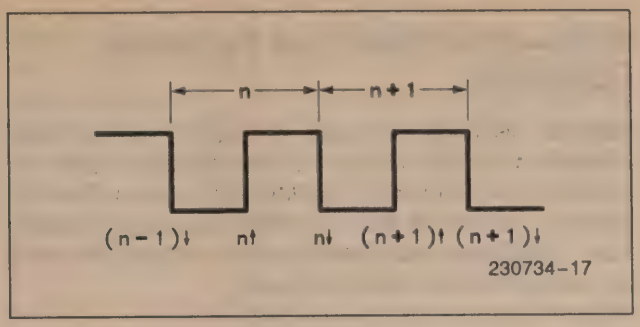
Figure 8B. 8208 Interface to iAPX 286

Configuration Charts

The 8208 operates in three basic configurations—C0, C1, C2—depending upon the programming of CFS (PD0), $\overline{RFS}$ (PD2), and $\overline{FFS}$ (PD7). Table 8 shows these configurations. These modes determine the clock edges for the 8208's programmable signals, as shown in Table 9. Finally, Table 10 gives the programmable AC parameters of the 8208 as a fuction of configuration. The non-programmable parameters are listed under AC Characteristics.

Using the Timing Charts

The notation used to indicate which clock edge triggers an output transition is "n↑" or "n↓", where "n" is the number of clock periods that have passed since clock 0, the reference clock, and "↑" refers to rising edge and "↓" to falling edge. A clock period is defined as the interval from a clock falling edge to the following falling edge. Clock edges are defined as shown below.



The clock edges which trigger transitions on each 8208 output are tabulated in Table 9. "H" refers to the high-going transition, and "L" to low-going transition.

Clock 0 is defined as the clock in which the 8208 begins a memory cycle, either as a result of a port request which has just arrived, or of a port request which was stored previously but could not be serviced at the time of its arrival because the 8208 was performing another memory cycle. Clock 0 is identified externally by the leading edge of RAS, which is always triggered on 0.

Table 8. 8208 Configurations

Timing Conf.	CFS(PD0)	RFS(PD2)	FFS(PD7)	Wait States*
C_0	iAPX286(1)	FAST RAM(1)	16 MHz(1)	0
C_1	iAPX286(1)	SLOW RAM(0)	16 MHz(1)	1
C_0	iAPX286(1)	FAST RAM(1)	12 MHz (0)	0
C_0	iAPX286(1)	SLOW RAM(0)	12 MHz (0)	0
C_2	iAPX186(0)	DON'T CARE	DON'T CARE	0

* Using $\overline{EAACK}$ (synchronous mode)

Table 9. Timing Chart

		$\overline{RAS}$		ADDRESS		$\overline{CAS}$		WE		$\overline{EAACK}$		$\overline{LAACK}$		$\overline{XACK}$	
Cn	Cycle	L	H	Col	Row	L	H	H	L	L	H	L	H	L	H
0	RD,RF	0↓	3↓	0↓	2↓	1↓	4↓			1↓	4↓	2↓	5↓	3↓	RD
	WR	0↓	5↓	0↓	3↓	2↓	5↓	1↓	5↓	1↓	4↓	1↓	4↓	3↓	WR
1	RD,RF	0↓	4↓	0↓	3↓	1↓	6↓			2↓	5↓	2↓	5↓	4↓	RD
	WR	0↓	5↓	0↓	3↓	2↓	5↓	1↓	5↓	1↓	4↓	1↓	4↓	3↓	WR
2	RD,RF	0↓	2↓	0↓	2↓	0↓	3↓			0↓	2↓	1↓	3↓	2↓	RD
	WR	0↓	4↓	0↓	3↓	1↓	4↓	0↓	4↓	0↓	2↓	1↓	3↓	2↓	WR

NOTES FOR INTERPRETING THE TIMING CHART:

1. COLUMN ADDRESS is the time column address becomes valid.
2. The $\overline{CAS}$, $\overline{EAACK}$, $\overline{LAACK}$ and $\overline{XACK}$ outputs are not issued during refresh.
3. $\overline{XACK}$—high is reset asynchronously by command going inactive and not by a clock edge.
4. $\overline{EAACK}$ is used in synchronous mode, $\overline{LAACK}$ and $\overline{XACK}$ in asynchronous mode.
5. ADDRESS-Row is the clock edge where the 8208 A0 switches from current column address to the next row address.
6. If a cycle is inhibited by PCTL = 1 (Multibus I/F mode) then $\overline{CAS}$ is not activated during write cycle and $\overline{XACK}$ is not activated in either read or write cycles.

8208—DRAM Interface Parameter Equations

Several DRAM parameters, but not all, are a direct function of 8208 timings, and the equations for these parameters are given in the following tables. The following is a list of those DRAM parameters which have NOT been included in the following tables, with an explanation for their exclusion.

WRITE CYCLE

tDS: system-dependent parameter.
tDH: system-dependent parameter.
tDHR: system-dependent parameter.

READ, WRITE REFRESH CYCLES

tRAC: response parameter.
tCAC: response parameter.
tREF: See "Refresh Period Options".
tCRP: must be met only if CAS-only cycles, which do not occur with 8208, exist.
tRAH: See "A.C. Characteristics"
tRCD: See "A.C. Characteristics"
tASC: See "A.C. Characteristics"
tASR: See "A.C. Characteristics"
tOFF: response parameter.

Table 10. Programmable Timings

Read and Refresh Cycles

Parameter	C2-Slow Cycle	C0-Fast Cycle	C1-Fast Cycle	Notes
tRP	2TCLCL-T25	3TCLCL-T25	3TCLCL-T25	1
tCPN	1.5TCLCL-T34	3TCLCL-T34	2TCLCL-T34	1
tRSH	2TCLCL-T32	2TCLCL-T33	3TCLCL-T33	1
tCSH	3TCLCL-T25	4TCLCL-T25	6TCLCL-T25	1
tCAH	2TCLCL-T32	TCLCL-T33	2TCLCL-T33	1
tAR	2TCLCL-T25	2TCLCL-T25	3TCLCL-T25	1
tT	3/30	3/30	3/30	2
tRC	4TCLCL	6TCLCL	7TCLCL	1
tRAS	2TCLCL-T25	3TCLCL-T25	4TCLCL-T25	1
tCAS	3TCLCL-T32	3TCLCL-T33	5TCLCL-T33	1
tRCS	1.5TCLCL-TCL-T36-TBUF	2TCLCL-TCL-T36-TBUF	2TCLCL-TCL-T36-TBUF	1
tRCH	TCLCL-T32 & T36 MIN	TCLCL-T32	TCLCL-T32	1

Write Cycles

Parameter	C2-Slow Cycle	C0-Fast Cycle	C1-Fast Cycle	Notes
tRP	2TCLCL-T25	3TCLCL-T25	3TCLCL-T25	1
tCPN	2.5TCLCL-T34	4TCLCL-T34	4TCLCL-T34	1
tRSH	3TCLCL-T32	3TCLCL-T33	3TCLCL-T33	1
tCSH	4TCLCL-T25	5TCLCL-T25	5TCLCL-T25	1
tCAH	2TCLCL-T32	TCLCL-T33	TCLCL-T33	1
tAR	3TCLCL-T25	3TCLCL-T25	3TCLCL-T25	1
tT	3/30	3/30	3/30	2
tRC	6TCLCL	8TCLCL	8TCLCL	1
tRAS	4TCLCL-T25	5TCLCL-T25	5TCLCL-T25	1
tCAS	3TCLCL-T32	3TCLCL-T33	3TCLCL-T33	1
tWCH	3TCLCL-T32	3TCLCL-T33	3TCLCL-T33	1,3
tWCR	4TCLCL-T25	5TCLCL-T25	5TCLCL-T25	1,3
tWP	4TCLCL-T36-TBUF	4TCLCL-T36-TBUF	4TCLCL-T36-TBUF	1
tRWL	4TCLCL-T36-TBUF	4TCLCL-T36-TBUF	4TCLCL-T36-TBUF	1
tCWL	4TCLCL-T36-TBUF	4TCLCL-T36-TBUF	4TCLCL-T36-TBUF	1
tWCS	TCLCL-T36-TBUF	TCLCL-T36-TBUF	TCLCL-T36-TBUF	1

NOTES:
1. Minimum.
2. Value on right is maximum; value on left is minimum.
3. Applies to the eight warm-up cycles during initialization.

ABSOLUTE MAXIMUM RATINGS

Ambient Temperature Under Bias	−0°C to +70°C
Storage Temperature	−65°C to +150°C
Voltage On Any Pin With Respect to Ground	−0.5V to +7V
Power Dissipation	1.7 Watts

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

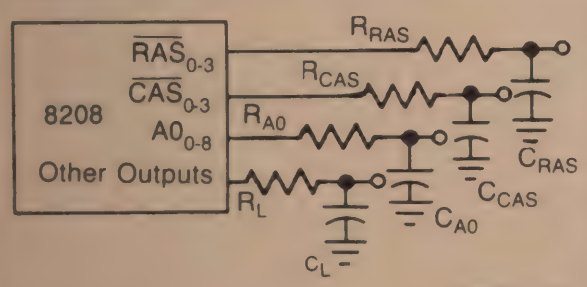
NOTICE: Specifications contained within the following tables are subject to change.

D.C. CHARACTERISTICS T_A = 0°C to 70°C; V_{CC} = 5.0V ±10%; V_{SS} = GND

Symbol	Parameter	Min	Max	Units	Comments
V_{IL}	Input Low Voltage	−0.5	+0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC} + 0.5	V	
V_{OL}	Output Low Voltage		0.45	V	Note 1
V_{OH}	Output High Voltage	2.4		V	Note 1
V_{ROL}	RAM Output Low Voltage		0.45	V	Note 1
V_{ROH}	RAM Output High Voltage	2.6		V	Note 1
I_{CC}	Supply Current		300	mA	T_A = 0°C
I_{LI}	Input Leakage Current		+10	μA	$0V \le V_{IN} \le V_{CC}$
V_{CL}	Clock Input Low Voltage	−0.5	+0.6	V	
V_{CH}	Clock Input High Voltage	3.8	V_{CC} + 0.5	V	
C_{IN}	Input Capacitance		20	pF	fc = 1 MHz

NOTE 1:
I_{OL} = 5 mA and I_{OH} = −0.32 mA (Typically I_{OL} = 10 mA)
WE: I_{OL} = 8 mA

A.C. Testing Load Circuit

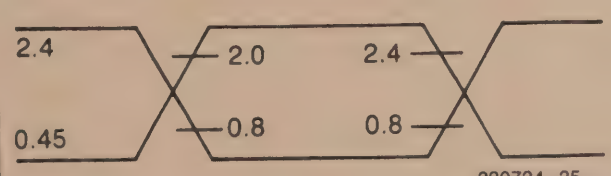


230734-18

R_{RAS} = 39Ω
R_{CAS} = 39Ω
R_{A0} = 22Ω
R_L = 33Ω

C_{RAS} = 150 pF
C_{CAS} = 150 pF
C_{A0} = 200 pF
C_L = 50 pF

A.C. Testing Input, Output Waveform



230734-25

A.C. Testing inputs (except clock) are driven at 2.4V for a logic "1" and 0.45V for a logic "0" (clock is driven at 4.0V and 0.45V for logic "1" and "0" respectively). Timing measurements are made at 2.0V, 2.4V for logic "1" and 0.8V for logic "0".

A.C. CHARACTERISTICS (Continued)

Ref	Symbol	Parameter	8208-16, 8208 (FFS=1)		8208-12, 8208-6 (FFS=0)		Units	Notes
			Min	Max	Min	Max		
RAM INTERFACE (Continued)								
38	TRWLTKH	$\overline{XACK}$ ↑ from $\overline{RD}$ ↑, $\overline{WR}$ ↑ Delay		50		55	ns	
39	TCLAKL	$\overline{AACK}$ ↓ from $\overline{CLK}$ ↓ Delay		35		35	ns	
40	TCLAKH	$\overline{AACK}$ ↑ from CLK ↓ Delay		50		60	ns	
REFRESH REQUEST								
41	TRFVCL	RFRQ to CLK ↓ Setup	20		30		ns	
42	TCLRFX	RFRQ to CLK ↓ Hold	10		10		ns	
43	TFRFH	Failsafe RFRQ Pulse Width	TCLCL + 30		TCLCL + 50		ns	19
44	TRFXCL	Single RFRQ Inactive to CLK ↓ Setup	20		30		ns	20
45	TBRFH	Burst RFRQ Pulse Width	2TCLCL + 30		2TCLCL + 50		ns	19

The following RC loading is assumed:

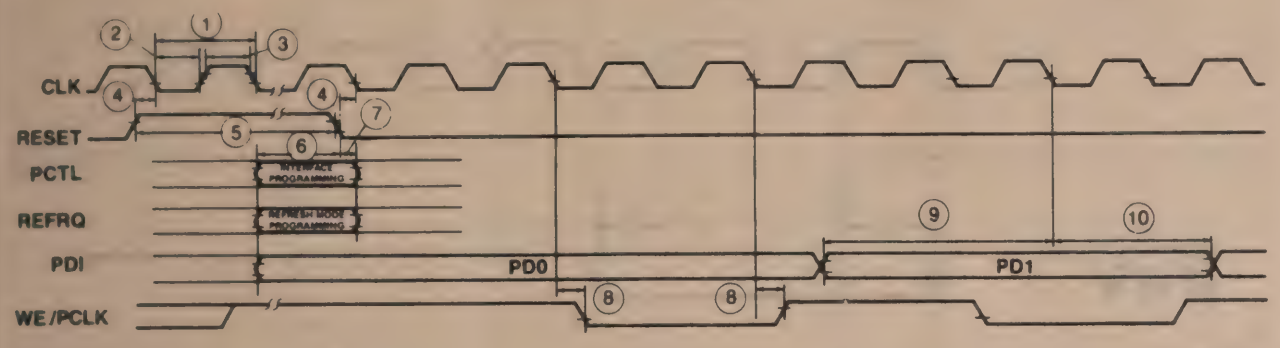
$A0_{0-8}$ R = 22Ω C = 200 pF
RAS_{0-1}, CAS_{0-1} R = 39Ω C = 150 pF
AACK, WE/PCLK R = 33Ω C = 50 pF

NOTES:

1. Specification when programmed in the Fast Cycle processor mode (iAPX 286 mode). 8208-16, -12 only.
2. Specification when programmed in the Slow Cycle processor mode (iAPX 186 mode). 8208-8, -6 only.
3. tR and tF are referenced from the 3.5V and 1.0V levels.
4. RESET is internally synchronized to CLK. Hence a set-up time is required only to guarantee its recognition at a particular clock edge.
5. The first programming bit (PD0) is also sampled by RESET going low.
6. TCLPDX is guaranteed if programming data is shifted using PCLK.
8. TRWVCL is not required for an asynchronous command except to guarantee its recognition at a particular clock edge.
9. Valid when programmed in either Fast or Slow Cycle mode.
10. tASR is a user specified parameter and its value should be added accordingly to TAVCL.
11. When programmed in Slow Cycle mode and 125 ns ≤ TCLCL < 200 ns.
12. When programmed in Slow Cycle mode and 200 ns ≤ TCLCL.
13. Specification for Test Load conditions.
14. tRCD (actual) = tRCD (specification) + 0.06 (ΔC_{RAS}) - 0.06(ΔC_{CAS}) where ΔC = C (test load) - C (actual) in pF. (These are first order approximations.)
15. tRAH (actual) = tRAH (specification) + 0.06 (ΔC_{RAS}) - 0.022 (ΔC_{A0}) where ΔC = C (test load) - C (actual) in pF. (These are first order approximations.)
16. tASR (actual) = tASR (specification) + 0.06 (ΔC_{A0}) - 0.025 (ΔC_{RAS}) where ΔC = C (test load) - C (actual) in pF. (These are first order approximations.)
17. tASC (actual) = tASC (specification) + 0.06 (ΔC_{A0}) - 0.025 (ΔC_{CAS}) where ΔC (test load) - C (actual) in pF. (These are first order approximations.)
18. tASC is a function of clock frequency and thus varies with changes in frequency. A minimum value is specified.
19. TFRFH and TBRFH pertain to asynchronous operation only.
20. Single RFRQ should be supplied synchronously to avoid burst refresh.

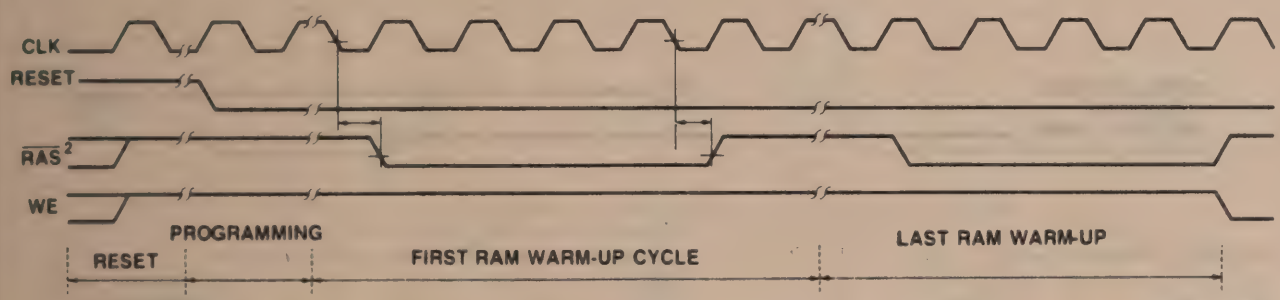
WAVEFORMS

Clock and Programming Timings



230734-19

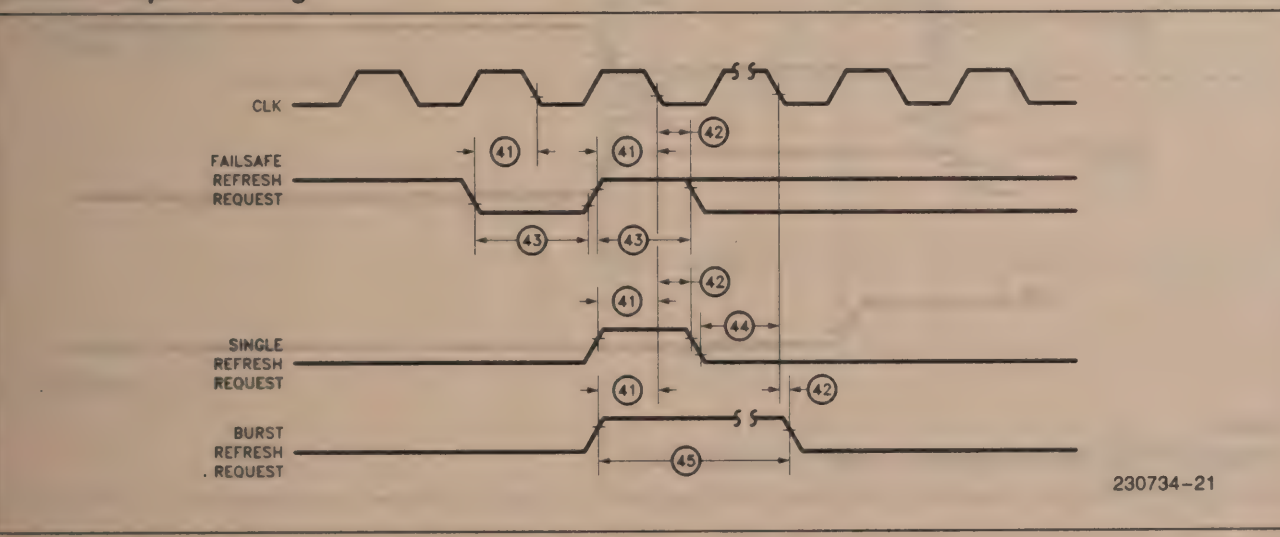
RAM Warm-up Cycles



230734-20

NOTE:
The present example assumes a $\overline{RAS}$ four clocks long.

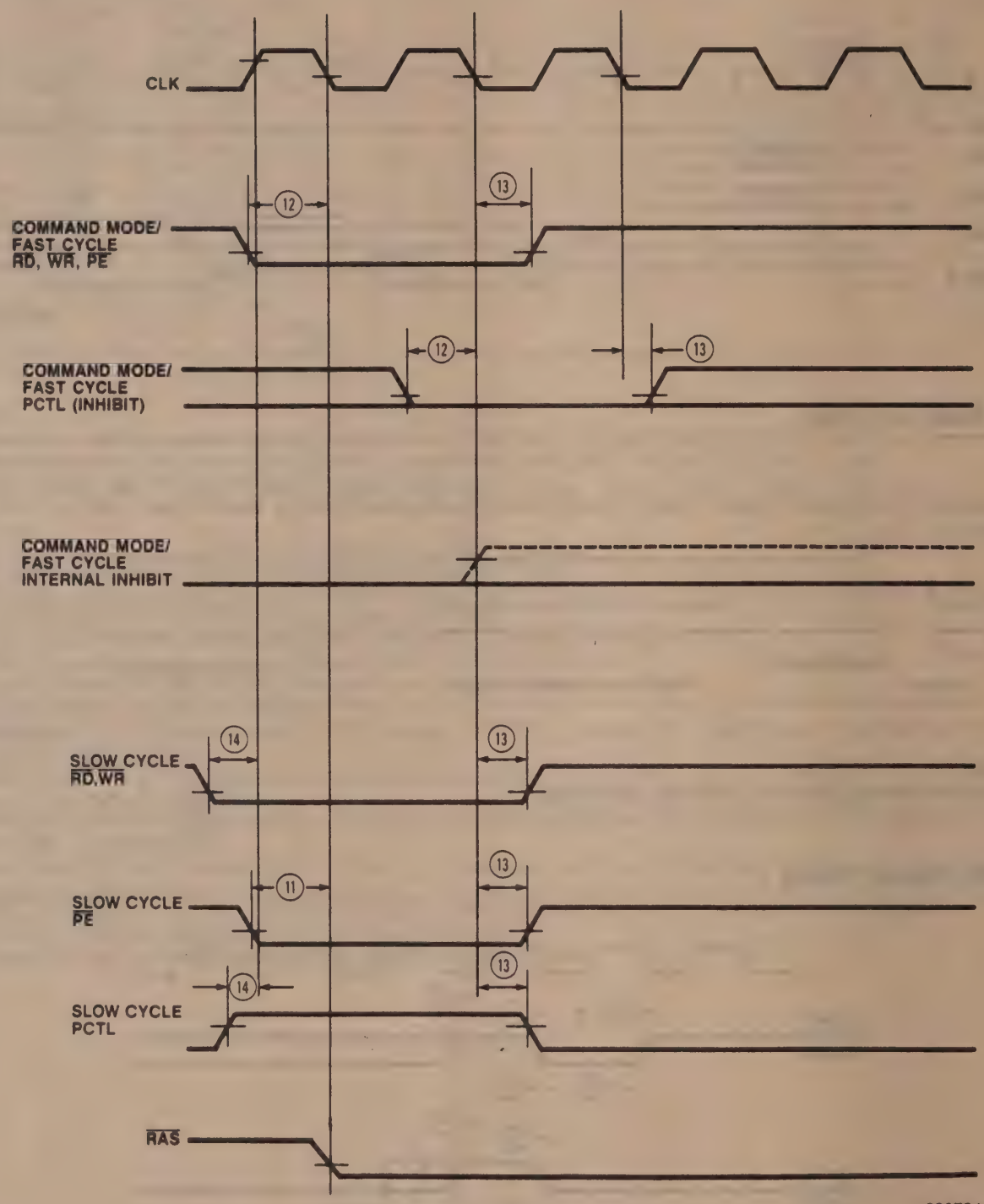
Refresh Request Timing



230734-21

WAVEFORMS (Continued)

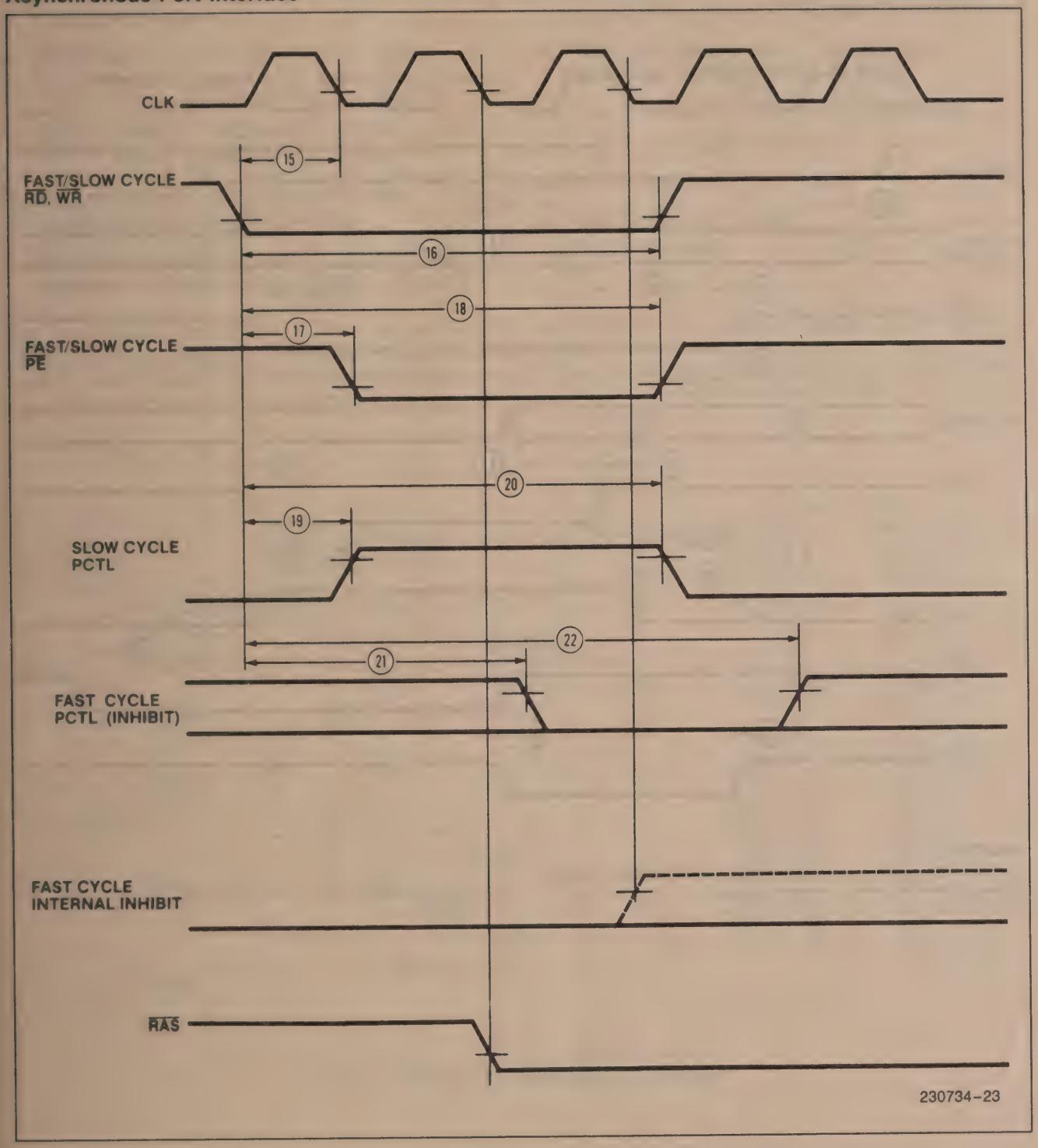
Synchronous Port Interface



NOTE:
Actual transitions are programmable. Refer to Tables 8 and 9.

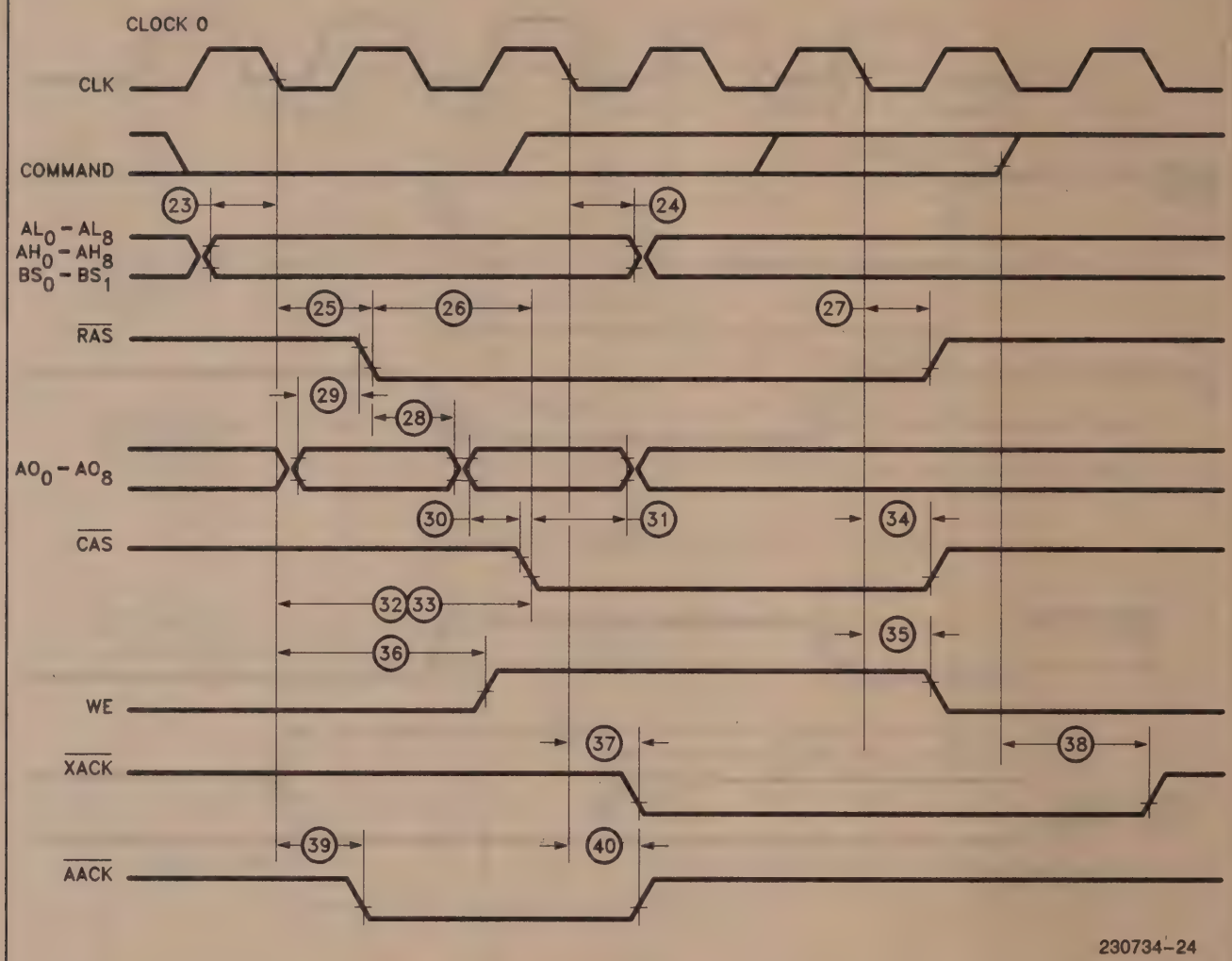
WAVEFORMS (Continued)

Asynchronous Port Interface



WAVEFORMS (Continued)

RAM Interface Timing

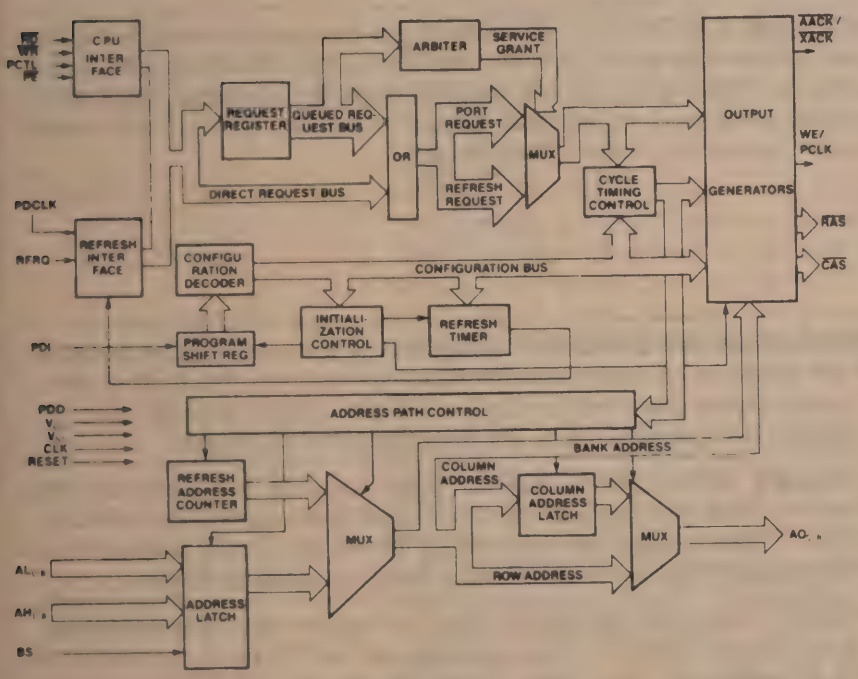


NOTE:
Actual transitions are programmable. See Tables 8 and 9.

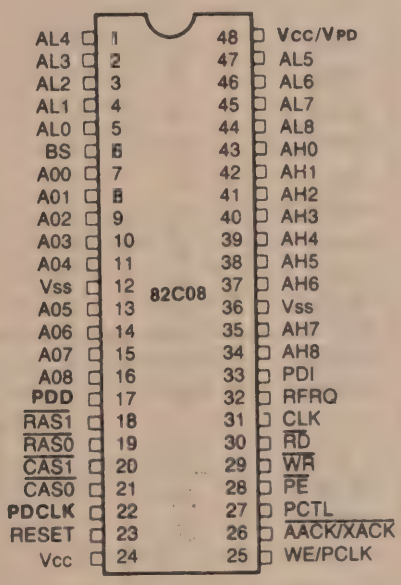
82C08
CHMOS DYNAMIC RAM CONTROLLER

- 0 Wait State with INTEL Microprocessors
- iAPX 286 (10, 8, 6 MHz) } 82C08-20 4–20 MHz, 82C08-16 4–16 MHz, 82C08-12 4–12 MHz
  iAPX 186/88, 86/88 } 82C08-10 2–10 MHz, 82C08-8 2–8 MHz
- Supports 64K and 256K DRAMS. Optimized for CMOS DRAMs
- Power Down Mode with Programmable Memory Refresh using Battery Backup
- Directly Addresses and Drives up to 1 Megabyte without External Drivers
- Microprocessor Data Transfer and Advance Acknowledge Signals
- Five Programmable Refresh Modes
- Automatic RAM Warm-up
- Pin-Compatible with 8208
- Plastic DIP 48 Lead
  PLCC 68 Lead

The Intel 82C08 Dynamic RAM Controller is a CMOS, high performance, systems oriented, Dynamic RAM controller that is designed to easily interface 64K and 256K Dynamic RAMs to Intel and other microprocessors. The 82C08 also has a power down mode where only the refresh logic is activated using battery backup.



231357–1



231357–2

Figure 1. Block Diagram and Pinout Diagram

Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied. Information contained herein supersedes previously published specifications on these devices from Intel.
December 1985
Order Number: 231357-003
© Intel Corporation, 1985

Table 1. Pin Description

Symbol	Pin	Type	Name and Function
AL0 AL1 AL2 AL3 AL4 AL5 AL6 AL7 AL8	5 4 3 2 1 47 46 45 44	I I I I I I I I I	**ADDRESS LOW:** These lower order address inputs are used to generate the column address for the internal address multiplexer. In iAPX 286 mode (CFS = 1), these addresses are latched internally.
AH0 AH1 AH2 AH3 AH4 AH5 AH6 AH7 AH8	43 42 41 40 39 38 37 35 34	I I I I I I I I I	**ADDRESS HIGH:** These higher order address inputs are used to generate the row address for the internal address multiplexer. In iAPX 286 mode, these addresses are latched internally.
BS	6	I	**BANK SELECT:** This input is used to select one of the two banks of the dynamic RAM array.
AO0 AO1 AO2 AO3 AO4 AO5 AO6 AO7 AO8	7 8 9 10 11 12 13 14 15	O O O O O O O O O	**ADDRESS OUTPUTS:** These outputs are designed to provide the row and column addresses, of either the CPU or the refresh counter, to the dynamic RAM array. These outputs drive the dynamic RAM array directly and need no external drivers. However, they typically need series resistors to match impedances.
$\overline{RAS0}$ $\overline{RAS1}$	19 18	O O	**ROW ADDRESS STROBE:** These outputs are used by the dynamic RAM array to latch the row address, present on the AO0–8 pins. These outputs are selected by the BS pin. These outputs drive the dynamic RAM array directly and need no external drivers.
$\overline{CAS0}$ $\overline{CAS1}$	21 20	O O	**COLUMN ADDRESS STROBE:** These outputs are used by the dynamic RAM array to latch the column address, present on the AO0–8 pins, These outputs are selected by the BS pin. These outputs drive the dynamic RAM array directly and need no external drivers.
RESET	23	I	**RESET:** This active high signal causes all internal counters to be reset. Upon release of RESET, data appearing at the PDI pin is clocked-in by the PCLK output. The states of the PDI, PCTL, and RFRQ pins are sampled by RESET going inactive and are used to program the 82C08. An 8-cycle dynamic RAM warm-up is performed after clocking PDI bits into the 82C08.
WE/ PCLK	25	O	**WRITE ENABLE/PROGRAMMING CLOCK:** Immediately after a RESET this pin becomes PCLK and is used to clock serial programming data into the PDI pin. After the 82C08 is programmed this active high signal provides the dynamic RAM array the write enable input for a write operation.

Symbol	Pin	Type	Name and Function
$\overline{AACK}$/ $\overline{XACK}$	26	O	**ADVANCE ACKNOWLEDGE/TRANSFER ACKNOWLEDGE:** When the X programming bit is set to logic 0 this pin is $\overline{AACK}$ and indicates that the processor may continue processing and that data will be available when required. This signal is optimized for the system by programming the S program-bit for synchronous or asynchronous operation. The S programming bit determines whether this strobe will be early or late. If another dynamic RAM cycle is in progress at the time of the new request, the $\overline{AACK}$ is delayed. When the X programming bit is set to logic 1 this pin is $\overline{XACK}$ and indicates that data on the bus is valid during a read cycle or that data may be removed from the bus during a write cycle. $\overline{XACK}$ is a MULTIBUS compatible signal.
PCTL	27	I	**PORT CONTROL:** This pin is sampled on the falling edge of RESET. It configures the 82C08 to accept command inputs or processor status inputs. If PCTL is low after RESET the 82C08 is programmed to accept bus/ multibus command inputs or iAPX 286 status inputs. If PCTL is high after RESET the 82C08 is programmed to accept status inputs from iAPX 86 or iAPX 186 type processors. The S2 status line should be connected to this input if programmed to accept iAPX 86 or iAPX 186 inputs. When programmed to accept bus commands or iAPX 286 status inputs, it should be tied low or it may be connected to INHIBIT when operating with MULTIBUS.
$\overline{PE}$	28	I	**PORT ENABLE:** This pin serves to enable a RAM cycle request. It is generally decoded from the address bus.
$\overline{WR}$	29	I	**WRITE:** This pin is the write memory request command input. This input also directly accepts the $\overline{S0}$ status line from Intel processors.
$\overline{RD}$	30	I	**READ:** This pin is the read memory request command pin. This input also directly accepts the $\overline{S1}$ status line from Intel processors.
CLK	31	I	**CLOCK:** This input provides the basic timing for sequencing the internal logic.
RFRQ	32	I	**REFRESH REQUEST:** This input is sampled on the falling edge of RESET. If RFRQ is high at RESET then the 82C08 is programmed for internal-refresh request or external-refresh request with failsafe protection. If RFRQ is low at RESET then the 82C08 is programmed for external-refresh without failsafe protection or burst refresh. Once programmed the RFRQ pin accepts signals to start an external-refresh with failsafe protection or external-refresh without failsafe protection or a burst refresh. RFRQ is also sampled when PDD is activated. When RFRQ = 1 it will cause 3 burst refresh cycles.
PDI	33	I	**PROGRAM DATA INPUT:** This input is sampled by RESET going low. It programs the various user selectable options in the 82C08. The PCLK pin shifts programming data into the PDI input from an external shift register. This pin may be strapped low to a default iAPX 186 mode configuration or high to a default iAPX 286 mode configuration.
*PDD	17	I	**POWER DOWN DETECT:** This input is sampled before every memory cycle to inform the 82C08 of system detection of power failure. When active, the 82C08 remains in power down mode and performs memory refresh only ($\overline{RAS}$-only refresh). In power down mode the 82C08 uses PDCLK for timing and VPD for power.
*PDCLK	22	I	**POWER DOWN CLOCK:** This pin is used as a clock for internal refresh circuits during power down. The input can be asynchronous to pin 31. Extended refresh is achieved by slowing down this clock. This pin should be grounded if not used.
*V_{CC}/V_{PD}	48	I	**POWER:** Power supply for internal logic. This should be held active during power down.
V_{CC}	24	I	**POWER:** Supply for drivers. Need not be held active during power down.
V_{SS}	12 36	I I	GROUND GROUND

*Different function than the HMOS 8208.

GENERAL DESCRIPTION

The Intel 82C08 Dynamic RAM Controller is a microcomputer peripheral device which provides the necessary signals to address, refresh, and directly drive 64K and 256K dynamic RAMs.

The 82C08 supports several microprocessor interface options including synchronous and asynchronous operations for iAPX 86, iAPX 186, iAPX 286, and MULTIBUS. The 82C08 will also interface to non-Intel microprocessors.

The 82C08 is a CHMOS version of the 8208 and is pin compatible with it. Three pins—17, 22, and 48—of the 82C08 are different from the 8208. They provide a power down mode that allows the system to run at a much lower ICC. In this mode, the 82C08 refreshes the DRAM using battery backup. The power down current (I_{PD}) that is drawn by the 82C08 is very small compared to the I_{CC} which allows memory to be kept alive with a battery. A separate refresh clock, pin 22, allows the designer to take advantage of RAMs that permit extended memory refresh.

The 82C08 also has some timing changes versus the 8208. In order to eliminate the external bus latches, both WE and $\overline{CAS}$ timings are shortened. These timing changes are backwards-compatible for 8208 designs.

FUNCTIONAL DESCRIPTION

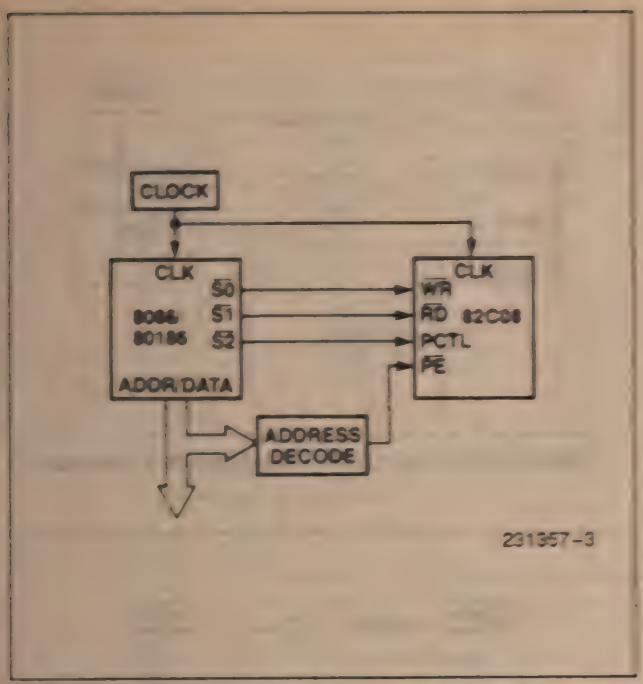
Processor Interface

The 82C08 has control circuitry capable of supporting one of several possible bus structures. The 82C08 may be programmed to run synchronous or asynchronous to the processor clock. The 82C08 has been optimized to run synchronously with Intel's iAPX 86, iAPX 88, iAPX 186/188 and iAPX 286. When the 82C08 is programmed to run in asynchronous mode, the 82C08 inserts the necessary synchronization circuitry for the $\overline{RD}$, $\overline{WR}$ inputs.

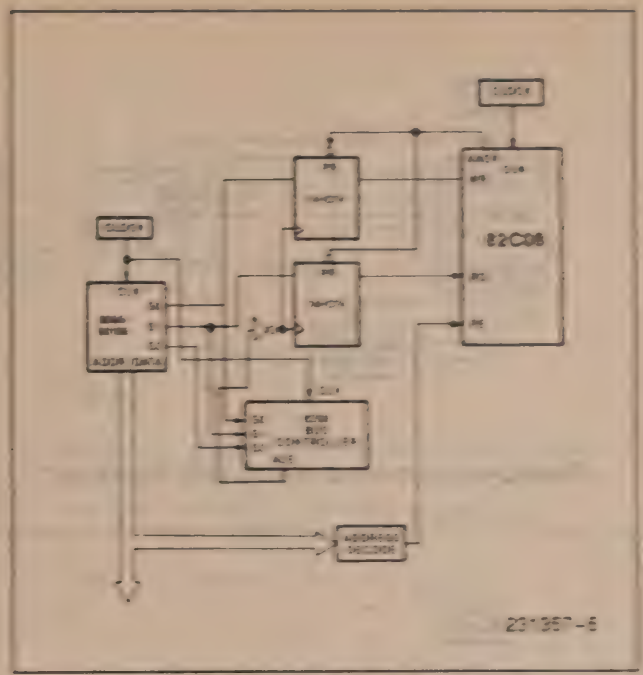
The 82C08 achieves high performance (i.e. no wait states) by decoding the status lines directly from the processor. The 82C08 can also be programmed to receive read or write MULTIBUS commands or commands from a bus controller.

The 82C08 may be programmed to operate synchronously to the processor. It can also be programmed to run at various frequencies. (See Microprocessor Clock Frequency Option.)

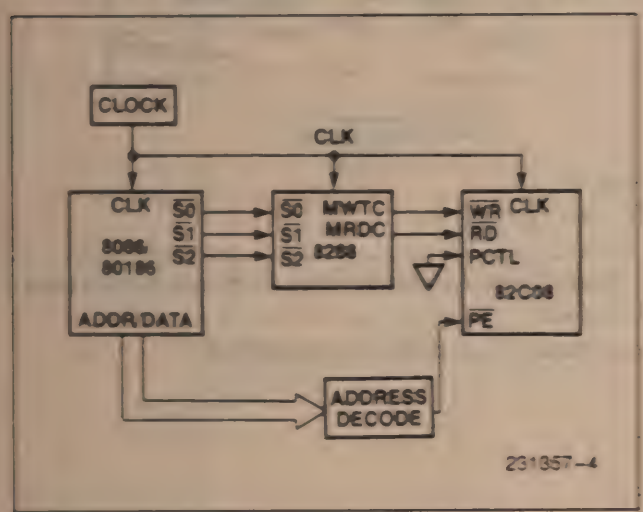
Figure 2 shows the different processor interfaces to the 82C08 using the synchronous or asynchronous mode and status or command interface. Figure 3 shows detailed interfaces to the iAPX 186 and iAPX 286 processors.



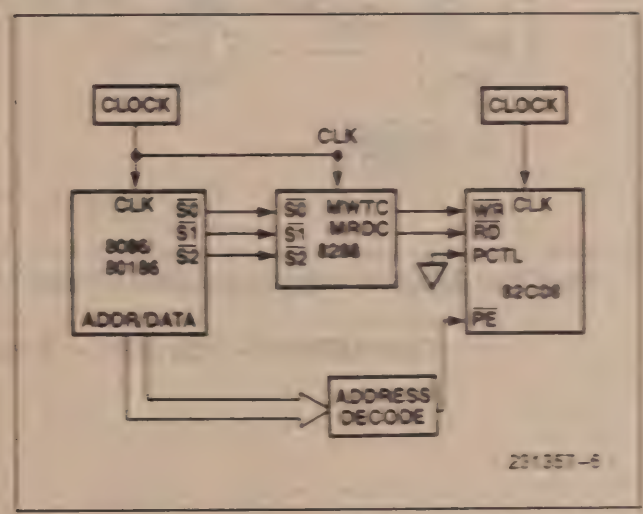
Slow-Cycle Synchronous-Status Interface



Slow-Cycle Asynchronous-Status Interface

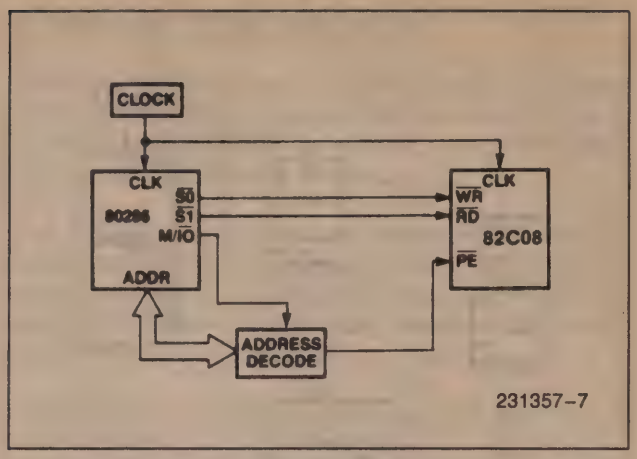


Slow-Cycle Synchronous-Command Interface

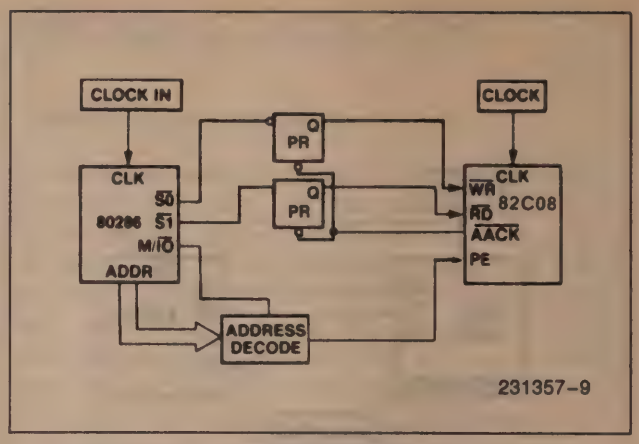


Slow-Cycle Asynchronous-Command Interface

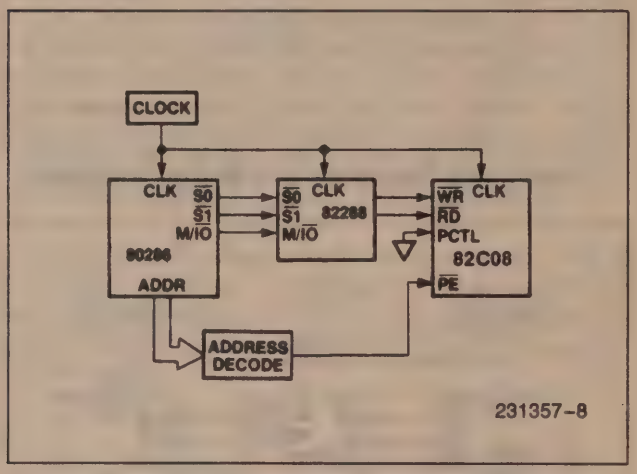
Figure 2A. Slow-cycle (CFS = 0) Port Interfaces Supported by the 82C08



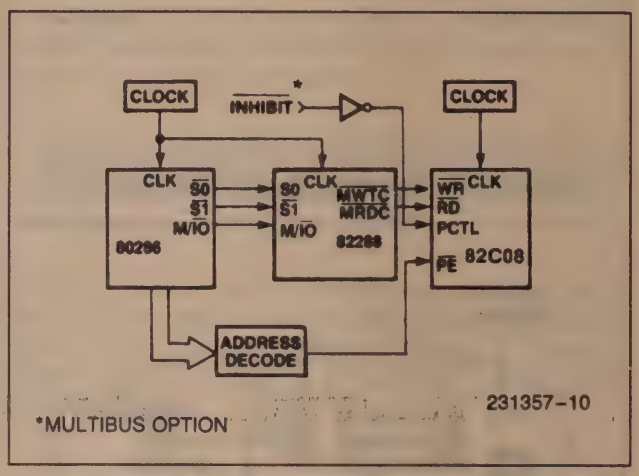
Fast-Cycle Synchronous-Status Interface



Fast-Cycle Asynchronous-Status Interface



Fast-Cycle Synchronous-Command Interface



Fast-Cycle Asynchronous-Command Interface

Figure 2B. Fast-cycle (CFS = 1) Port Interfaces Supported by the 82C08

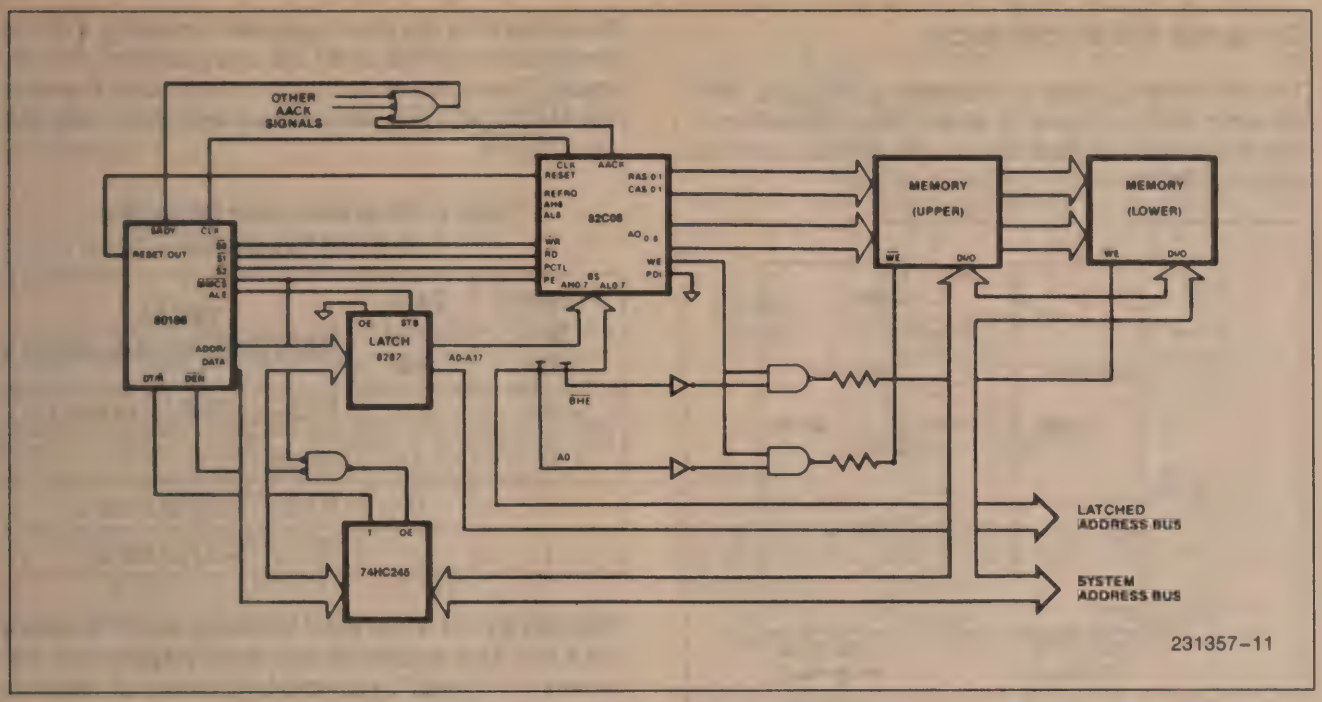


Figure 3A. 82C08 Interface to an 80186

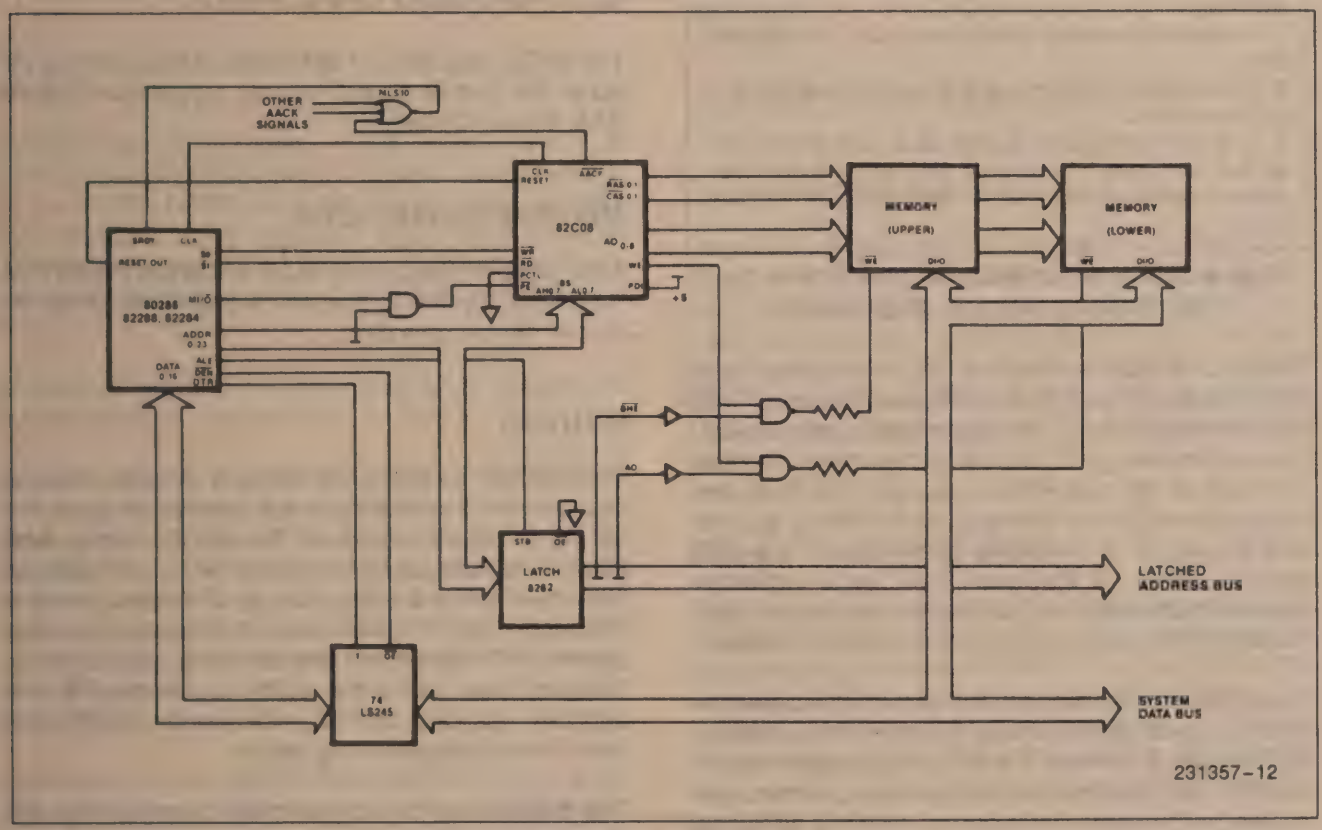
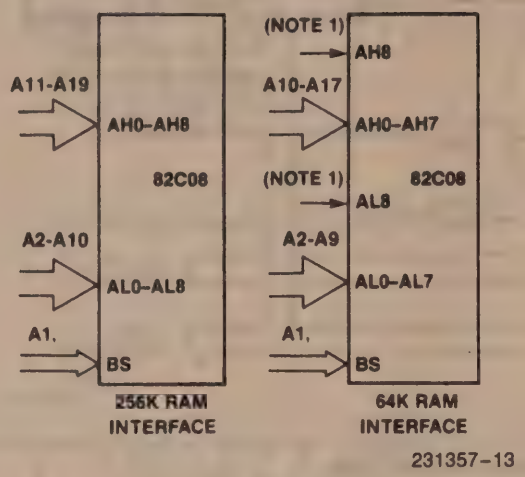


Figure 3B. 82C08 Interface to an 80286

Dynamic RAM Interface

The 82C08 is capable of addressing 64K and 256K dynamic RAMs. Figure 3 shows the connection of the processor address bus to the 82C08 using the different RAMs.



NOTES:
1. Unassigned address input pins should be strapped high.
2. A0 along with BHE are used to select a byte within a processor word.
3. Low order address bit is used as a bank select input so that consecutive memory access requests are to alternate banks allowing bank interleaving of memory cycles.

Figure 3. Processor Address Interface to the 82C08 Using 64K, and 256K RAMS

The 82C08 divides memory into two banks, each bank having its own Row ($\overline{RAS}$) and Column ($\overline{CAS}$) Address Strobe pair. This organization permits RAM cycle interleaving. RAM cycle interleaving overlaps the start of the next RAM cycle with the RAM precharge period of the previous cycle. Hiding the precharge period of one RAM cycle behind the data access period of the next RAM cycle optimizes memory bandwidth and is effective as long as successive RAM cycles occur in the alternate banks.

Successive data access to the same bank cause the 82C08 to wait for the precharge time of the previous RAM cycle. But when the 82C08 is programmed in an iAPX 186 synchronous configuration, consecutive cycles to the same bank do not result in additional wait states (i.e. 0 wait state).

If not all RAM banks are occupied, the 82C08 can be programmed to reassign the $\overline{RAS}$ and $\overline{CAS}$ strobes to allow using wider data words without increasing the loading on the $\overline{RAS}$ and $\overline{CAS}$ drivers.

Table 2 shows the bank selection decoding and the corresponding $\overline{RAS}$ and $\overline{CAS}$ assignments. For example, if only one RAM bank is occupied, then the two $\overline{RAS}$ and $\overline{CAS}$ strobes are activated with the same timing.

Table 2. Bank Selection Decoding and Word Expansion

Program Bit RB	Bank Input BS	82C08 $\overline{RAS}/\overline{CAS}$ Pair Allocation
0	0	$\overline{RAS}_{0,1}$, $\overline{CAS}_{0,1}$ to Bank 0
0	1	Illegal
1	0	$\overline{RAS}_0$, $\overline{CAS}_0$ to Bank 0
1	1	$\overline{RAS}_1$, $\overline{CAS}_1$ to Bank 1

Program bit RB is not used to check the bank select input BS. The system design must protect from accesses to "illegal", non-existent banks of memory by deactivating the PE input when addressing an "illegal", non-existent bank of memory.

The 82C08 adjusts and optimizes internal timings for either the fast or slow RAMs as programmed. (See RAM Speed Option.)

Memory Initialization

After programming, the 82C08 performs eight RAM "wake-up" cycles to prepare the dynamic RAM for proper device operation.

Refresh

The 82C08 provides an internal refresh interval counter and a refresh address counter to allow the 82C08 to refresh memory. The 82C08 has a 9-bit internal refresh address counter which will refresh 128 rows every 2 milliseconds, 256 rows every 4 milliseconds or 512 rows every 8 milliseconds, which allows all RAM refresh options to be supported. In addition, there exists the ability to refresh 256 row address locations every 2 milliseconds via the Refresh Period programming option.

The 82C08 may be programmed for any of five different refresh options: Internal refresh only, External refresh with failsafe protection, External refresh without failsafe protection, Burst refresh modes, or no refresh. (See Refresh Options.)

It is possible to decrease the refresh time interval by 10%, 20% or 30%. This option allows the 82C08 to compensate for reduced clock frequencies. Note

that an additional 5% interval shortening is built-in in all refresh interval options to compensate for clock variations and non-immediate response to the internally generated refresh request. (See Refresh Period Options.)

External Refresh Requests after RESET

External refresh requests are not recognized by the 82C08 until after it is finished programming and preparing memory for access. Memory preparation includes 8 RAM cycles to prepare and ensure proper dynamic RAM operation. The time it takes for the 82C08 to recognize a request is shown below.

eg. 82C08 System Response:

TRESP = TPROG + TPREP

where: TPROG = (40) (TCLCL) programming time

TPREP = (8) (32) (TCLCL) RAM warm-up time

if TCLCL = 125 ns then TRESP = 37 μs

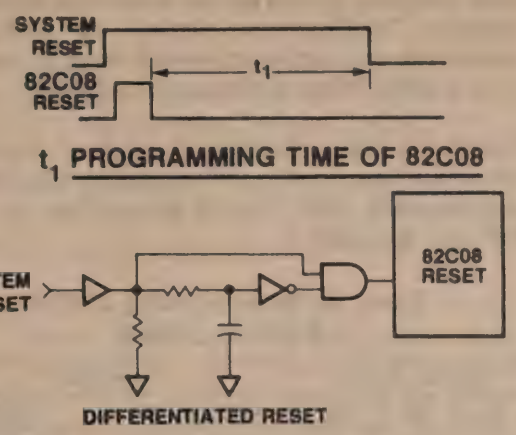
Reset

RESET is an asynchronous input, its falling edge is used by the 82C08 to directly sample the logic levels of the PCTL, RFRQ, and PDI inputs. The internally synchronized falling edge of reset is used to begin programming operations (shifting in the contents of the external shift register, if needed, into the PDI input).

Differentiated reset is unnecessary when the default synchronization programming is used.

Until programming is complete the 82C08 latches but does not respond to command or status inputs. A problem may occur if the S bit is programmed inconsistently from the Command which was latched before programming was completed. A simple means of preventing commands or status from occurring during this period is to differentiate the system reset pulse to obtain a smaller reset pulse for the 82C08.

The differentiated reset pulse would be shorter than the system reset pulse by at least the programming period required by the 82C08. The differentiated reset pulse first resets the 82C08, and system reset would reset the rest of the system. While the rest of the system is still in reset, the 82C08 completes its programming. Figure 4 illustrates a circuit to accomplish this task.



NOTES: 231357-14

1. Required only when the synchronization option is altered from its initial default value.
2. V_{CC} must be stable before system reset is activated when using this circuit.

Figure 4. 82C08 Differentiated Reset Circuit

Within four clocks after RESET goes active, all the 82C08 outputs will go high, except for AO0–2, which will go low.

OPERATIONAL DESCRIPTION

Programming the 82C08

The 82C08 is programmed after reset. On the falling edge of RESET, the logic states of several input pins are latched internally. The falling edge of RESET actually performs the latching, which means that the logic levels on these inputs must be stable prior to that time. The inputs whose logic levels are latched at the end of reset are the PCTL, RFRQ, and PDI pins.

Status/Command Mode

The processor port of the 82C08 is configured by the states of the PCTL pin. Which interface is selected depends on the state of the PCTL pin at the end of reset. If PCTL is high at the end of reset, the 8086/80186 Status interface is selected; if it is low, then the MULTIBUS or Command interface is selected.

The status lines of the 80286 are similar in code and timing to the Multibus command lines, while the status code and timing of the 8086 and 8088 are identical to those of the 80186 and 80188 (ignoring the differences in clock duty cycle). Thus there exists two interface configurations, one for the 80286 status or Multibus memory commands, which is called the Command interface, and one for 8086,

8088, 80186 or 80188 status, called the 8086 Status interface. The Command interface can also directly interface to the command lines of the bus controllers for the 8086, 8088, 80186 and the 80286.

The 80186 Status interface allows direct decoding of the status lines for the iAPX 86, iAPX 88, iAPX 186 and the iAPX 188. Table 3 shows how the status lines are decoded.

Table 3A. Status Coding of 8086, 80186 and 80286

Status Code			Function	
$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	8086/80186	80286*
0	0	0	INTERRUPT	INTERRUPT
0	0	1	I/O READ	I/O READ
0	1	0	I/O WRITE	I/O WRITE
0	1	1	HALT	IDLE
1	0	0	INSTRUCTION FETCH	HALT
1	0	1	MEMORY READ	MEMORY READ
1	1	0	MEMORY WRITE	MEMORY WRITE
1	1	1	IDLE	IDLE

* Refer to 80286 pin description table

Table 3B. 82C08 Response

82C08 Command			Function	
PCTL	$\overline{RD}$	$\overline{WR}$	8086/80186 Status Interface	80286 Status or Command Interface
0	0	0	IGNORE	IGNORE*
0	0	1	IGNORE	READ
0	1	0	IGNORE	WRITE
0	1	1	IGNORE	IGNORE
1	0	0	READ	IGNORE
1	0	1	READ	INHIBIT
1	1	0	WRITE	INHIBIT
1	1	1	IGNORE	IGNORE

*Illegal with CFS = 0

Refresh Options

Immediately after system reset, the state of the RFRQ input pin is examined. If RFRQ is high, the 82C08 provides the user with the choice between self-refresh and user-generated refresh with failsafe protection. Failsafe protection guarantees that if the user does not come back with another refresh request before the internal refresh interval counter times out, a refresh request will be automatically generated. If the RFRQ pin is low immediately after a reset, then the user has the choice of a single external refresh cycle without failsafe, burst refresh or no refresh.

Internal Refresh Only

For the 82C08 to generate internal refresh requests, it is necessary only to strap the RFRQ input pin high.

External Refresh with Failsafe

To allow user-generated refresh requests with failsafe protection, it is necessary to hold the RFRQ input high until after reset. Thereafter, a low-to-high transition on this input causes a refresh request to be generated and the internal refresh interval counter to be reset. A high-to-low transition has no effect on the 82C08. A refresh request is not recognized until a previous request has been serviced.

External Refresh without Failsafe

To generate single external refresh requests without failsafe protection, it is necessary to hold RFRQ low until after reset. Thereafter, bringing RFRQ high for one clock period will cause a refresh request to be generated. A refresh request is not recognized until a previous request has been serviced.

Burst Refresh

Burst refresh is implemented through the same procedure as a single external refresh without failsafe (i.e., RFRQ is kept low until after reset). Thereafter, bringing RFRQ high for at least two clock periods will cause a burst of up to 128 row address locations to be refreshed. A refresh request is not recognized until a previous request has been serviced (i.e. burst is completed).

No Refresh

It is necessary to hold RFRQ low until after reset. This is the same as programming External Refresh without Failsafe. No refresh is accomplished by keeping RFRQ low.

Option Program Data Word

PROGRAMMING FOR SLOW CYCLE

The program data word consists of 9 program data bits, PD0–PD8. If the first program data bit, PD0 is

set to logic 0, the 82C08 is configured to support iAPX 186, 188, 86, or 88 systems. The remaining bits, PD1–PD8, may then be programmed to optimize a selected system configuration. A default of all zeros in the remaining program bits optimizes the 82C08 timing for 8 MHz Intel CPUs using 150 ns (or faster) dynamic RAMs with no performance penalty.

PROGRAMMING FOR FAST CYCLE

If the first program data bit is set to logic 1, the 82C08 is configured to support iAPX 286 systems (Command mode). A default of all ones in the program bits optimizes the 82C08 timing for an 8 MHz 286 using 120 ns DRAMs at zero wait states. Note that the programming bits PD1–8 change polarity according to PD0. This ensures the same choice of options for both default modes.

Table 4A shows the various options that can be programmed into the 82C08.

Table 4A. Program Data Word

Program Data Bit	Name		Polarity/Function
	PD0 = 0	PD0 = 1	
PD0	CFS	CFS	CFS = 0 SLOW CYCLE CFS = 1 FAST CYCLE
PD1	$\overline{S}$	S	$\overline{S}$ = 0 SYNCHRONOUS* $\overline{S}$ = 1 ASYNCHRONOUS
PD2	$\overline{RFS}$	RFS	$\overline{RFS}$ = 0 FAST RAM* $\overline{RFS}$ = 1 SLOW RAM
PD3	$\overline{RB}$	RB	RAM BANK OCCUPANCY SEE TABLE 2
PD4	CI1	$\overline{CI1}$	COUNT INTERVAL BIT 1; SEE TABLE 6
PD5	CI0	$\overline{CI0}$	COUNT INTERVAL BIT 0; SEE TABLE 6
PD6	$\overline{PLS}$	PLS	$\overline{PLS}$ = 0 LONG REFRESH PERIOD* $\overline{PLS}$ = 1 SHORT REFRESH PERIOD
PD7	$\overline{FFS}$	FFS	$\overline{FFS}$ = 0 FAST CPU FREQUENCY* $\overline{FFS}$ = 1 SLOW CPU FREQUENCY
PD8	X	$\overline{X}$	X = 0 $\overline{AACK}$* X = 1 $\overline{XACK}$

* Default in both modes

Using an External Shift Register

The 82C08 may be programmed by using an external shift register with asynchronous load capability such as a 74HC165. The reset pulse serves to parallel load the shift register and the 82C08 supplies the clocking signal (PCLK) to shift the data into the PDI programming pin. Figure 6 shows a sample circuit diagram of an external shift register circuit.

Serial data is shifted into the 82C08 via the PDI pin (33), and clock is provided by the WE/PCLK pin (25), which generates a total of 9 clock pulses.

WE/PCLK is a dual function pin. During programming, it serves to clock the external shift register, and after programming is completed, it reverts to the write enable RAM control output pin. As the pin changes state to provide the write enable signal to the dynamic RAM array, it continues to clock the shift register. This does not present a problem because data at the PDI pin is ignored after programming. Figure 7 illustrates the timing requirements of the shift register.

Default Programming Options

After reset, the 82C08 serially shifts in a program data word via the PDI pin. This pin may be strapped low or high, or connected to an external shift register. Strapping PDI low causes the 82C08 to default to the iAPX 186 system configuration, while high causes a default to the iAPX 286 configuration. Table 4B shows the characteristics of the default configuration for Fast Cycle (PDI = 1) and Slow Cycle (PDI = 0). If further system flexibility is needed, one external shift register, like a 74HC165, can be used to tailor the 82C08 to its operating environment.

Table 4B. Default Programming

Synchronous interface
Fast RAM (Note 1)
2 RAM banks occupied
128 row refresh in 2 ms; 256 in 4 ms, 512 in 8 ms
Fast processor clock frequency
Advanced ACK strobe

NOTE:
1. For iAPX 86/186 systems either slow or fast (150 or 100 ns) RAMS will run at 8 MHz with zero wait states.

Synchronous/Asynchronous Mode (S program bit)

The 82C08 may be configured to accept synchronous or asynchronous commands ($\overline{RD}$, $\overline{WR}$, PCTL) and Port Enable ($\overline{PE}$) via the S program bit. The state of the S programming bit determines whether the interface is synchronous or asynchronous.

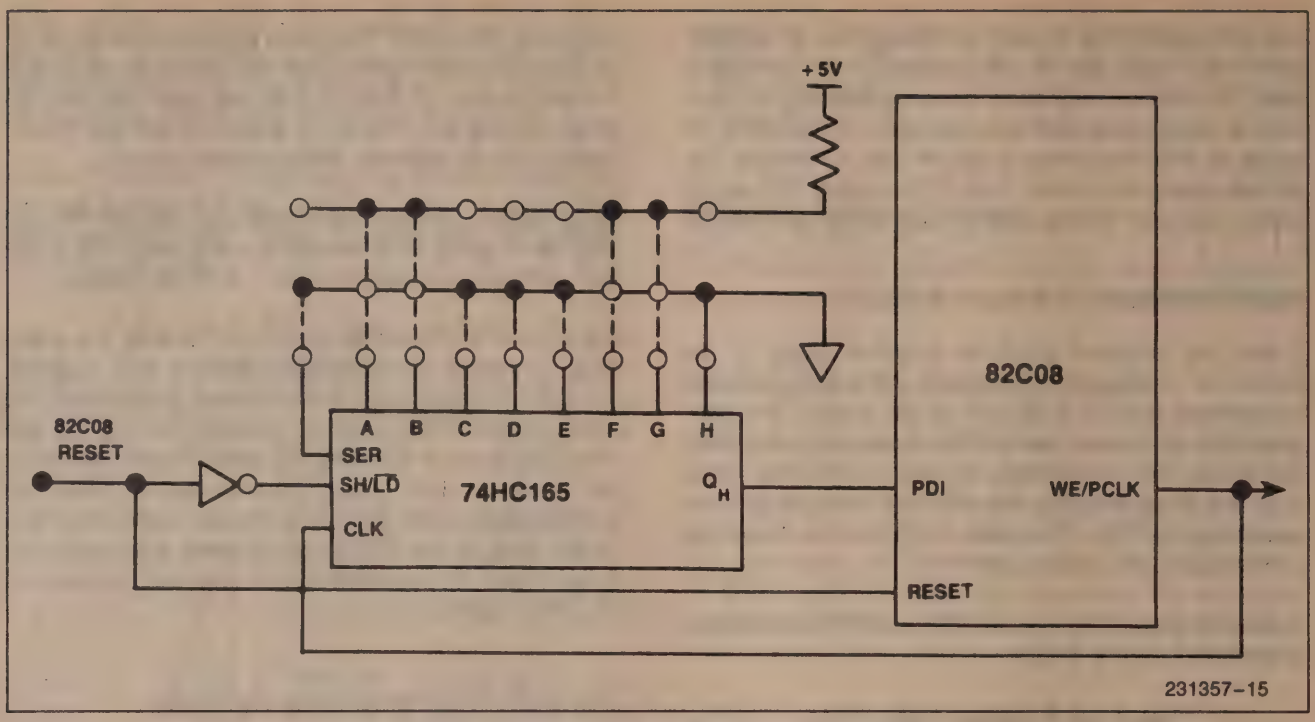
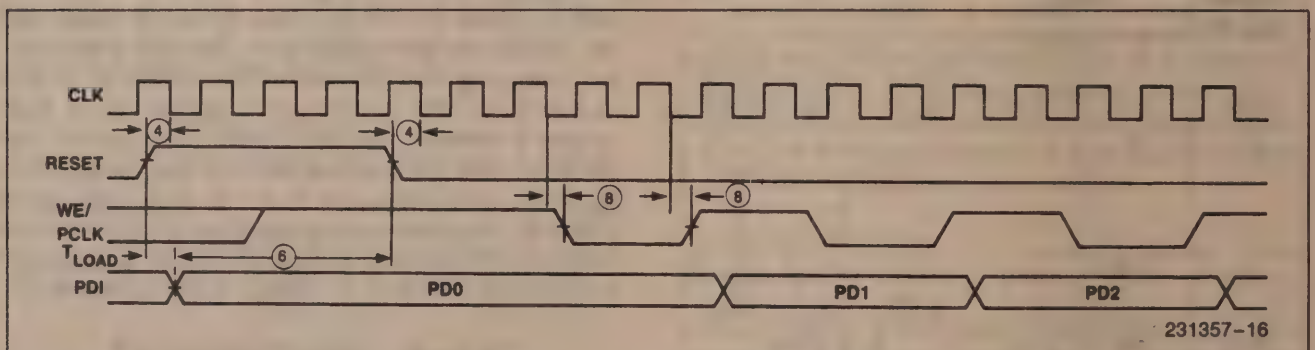


Figure 6. External Shift Register Interface



NOTES:
TRTVCL — Reset is an asychronous input, if reset occurs before TRTVCL, then it is guaranteed to be recognized.
TPGVCL — Minimum PDI valid time prior to reset going low.
TCLPC — MUX/PCLK delay.
TLOAD — Asychronous load data propagation delay.

Figure 7. Timing Illustrating External Shift Register Requirements for Programming the 82C08

While the 82C08 may be configured with either the Status or Command (MULTIBUS) interface in the Synchronous mode, certain restrictions exist in the Asynchronous mode. An Asynchronous-Command interface is directly supported. An Asynchronous-80186/80286 Status interface using the status lines of the 80186/80286 is supported with the use of TTL gates as illustrated in Figure 2. In the 80186 case, the TTL gates are needed to guarantee that status does not appear at the 82C08's inputs too much before address, so that a cycle would start before address was valid. In the case of the 80286, the TTL gates are used for lengthening the Status pulse, as required by the TRWL timing.

Microprocessor Clock Cycle Option (CFS and FFS program bits)

The 82C08 is programmed to interface with microprocessors with "slow cycle" timing like the 8086, 8088, 80186, and 80188, and with "fast cycle" microprocessors like the 80286. The CFS bit is used to select the appropriate timing.

The FFS option is used to select the speed of the microprocessor clock. Table 5 shows the various microprocessor clock frequency options that can be programmed. The external clock frequency must be

programmed so that the failsafe refresh repetition circuitry can adjust its internal timing accordingly to produce a refresh request as programmed.

Table 5. Microprocessor Clock Frequency Options

Program Bits		Processor	Clock Frequency
CFS	FFS		
0	0	iAPX 86, 88, 186, 188	≤ 5 MHz
0	1	iAPX 86, 88, 186, 188	≥ 6 MHz
1	0	iAPX 286	≤ 10 MHz
1	1	iAPX 286	≥ 12 MHz

RAM Speed Option (RFS program bit)

The RAM Speed programming option determines whether RAM timing will be optimized for a fast or slow RAM. Whether a RAM is fast or slow is measured relative to 100 ns DRAMs (fast) or 150 ns DRAMs (slow). This option is only a factor in Fast cycle Mode (CFS = 1).

Refresh Period Options (CI0, CI1 and PLS program bits)

The 82C08 refreshes with either 128 rows every 2 milliseconds, with 256 rows every 4 milliseconds or 512 rows every 8 milliseconds. This translates to one refresh cycle being executed approximately once every 15.6 microseconds. This rate can be changed to 256 rows every 2 milliseconds or a refresh approximately once every 7.8 microseconds via the Period Long/Short, program bit PLS, programming option.

The Count Interval 0 (CI0) and Count Interval 1 (CI1) programming options allow the rate at which refresh requests are generated to be increased in order to permit refresh requests to be generated close to the 15.6 or 7.8 microsecond period when the 82C08 is operating at reduced frequencies. The interval between refreshes is decreased by 0%, 10%, 20%, or 30% as a function of how the count interval bits are programmed. A 5% guardband is built-in to allow for any clock frequency variations. Table 6 shows the refresh period options available.

The numbers tabulated under Count Interval represent the number of clock periods between internal refresh requests. The percentages in parentheses represent the decrease in the interval between refresh requests.

Table 6. Refresh Count Interval Table

Ref. Period (μS)	CFS	PLS	FFS	Count Interval CI1, CI0 (82C08 Clock Periods)			
				00 (0%)	01 (10%)	10 (20%)	11 (30%)
15.6	1	1	1	236	212	188	164
7.8	1	0	1	118	106	94	82
15.6	1	1	0	148	132	116	100
7.8	1	0	0	74	66	58	50
15.6	0	1	1	118	106	94	82
7.8	0	0	1	59	53	47	41
15.6	0	1	0	74	66	58	50
7.8	0	0	0	37	33	29	25

The refresh count interval is set up for the following basic frequencies:

5 MHz slow cycle

8 MHz slow cycle

10 MHz fast cycle

16 MHz fast cycle

Example: Best 12 MHz fast cycle performance can be achieved using the basic frequency of 16 MHz (CFS = 1, FFS = 1) and the appropriate count interval bits (CI1 = 1, CI0 = 1) to reduce the frequency.

clock period × refresh count interval = refresh period

i.e. 83.3 ns × 164 = 13.6 μs

Example: 10 MHz slow cycle

CFS = 0, FFS = 1, CI1 = 0, CI0 = 0

i.e. 100 ns × 118 = 11.8 μs

Processor Timing

In order to run without wait states, $\overline{AACK}$ must be used and connected to the $\overline{SRDY}$ input of the appropriate bus controller. $\overline{AACK}$ is issued relative to a point within the RAM cycle and has no fixed relationship to the processors's request. The timing is such, however, that the processor will run without wait states, barring refresh cycles. In slow cycle, fast RAM configurations (8086, 80186), $\overline{AACK}$ is issued on the same clock cycle that issues $\overline{RAS}$.

Port Enable ($\overline{PE}$) set-up time requirements depend on whether the 82C08 is configured for synchronous

or asynchronous, fast or slow cycle operation. In a synchronous fast cycle configuration, $\overline{PE}$ is required to be set-up to the same clock edge as the commands. If $\overline{PE}$ is true (low), a RAM cycle is started; if not, the cycle is not started until the $\overline{RD}$ or $\overline{WR}$ line goes inactive and active again.

In asychronous operation, $\overline{PE}$ is required to be set-up to the same clock edge as the internally synchronized status or commands. Externally, this allows the internal synchronization delay to be added to the status (or command) -to-$\overline{PE}$ delay time, thus allowing for more external decode time than is available in synchronous operation.

The minimum synchronization delay is the additional amount that $\overline{PE}$ must be held valid. If $\overline{PE}$ is not held valid for the maximum synchronization delay time, it is possible that $\overline{PE}$ will go invalid prior to the status or command being synchronized. In such a case the 82C08 may not start a memory cycle. If a memory cycle intended for the 82C08 is not started, then no acknowledge ($\overline{AACK}$ or $\overline{XACK}$) is issued and the processor locks up in endless wait states.

Memory Acknowledge ($\overline{AACK}$, $\overline{XACK}$)

Two types of memory acknowledge signals are supplied by the 82C08. They are the Advanced Acknowledge strobe ($\overline{AACK}$) and the Transfer Acknowledge strobe ($\overline{XACK}$). The S programming bit optimizes $\overline{AACK}$ for synchronous operation ("early" $\overline{AACK}$) or asynchronous operation ("late" $\overline{AACK}$). Both the early and late $\overline{AACK}$ strobes are two clocks long for CFS = 0 and three clocks long for CFS = 1.

The $\overline{XACK}$ strobe is asserted when data is valid (for reads) or when data may be removed (for writes) and meets the MULTIBUS requirements. $\overline{XACK}$ is removed asynchronously by the command going inactive.

Since in an asynchronous operation the 82C08 removes read data before late $\overline{AACK}$ or $\overline{XACK}$ is recognized by the CPU, the user must provide for data latching in the system until the CPU reads the data. In synchronous operation data latching is unnecessary, since the 82C08 will not remove data until the CPU has read it.

If the X programming bit is high, the strobe is configured as $\overline{XACK}$, while if the bit is low, the strobe is configured as $\overline{AACK}$.

Data will always be valid a fixed time after the occurrence of the advanced acknowledge. Thus, the advanced acknowledge may also serve as a RAM cycle timing indicator.

General System Considerations

1. The $\overline{RAS}$0, 1, $\overline{CAS}$0, 1, and AO0–8 output buffers are designed to directly drive the heavy capacitive loads of the dynamic RAM arrays. To keep the RAM driver outputs from ringing excessively in the system environment it is necessary to match the output impedance with the RAM array by using series resistors. Each application may have different impedance characteristics and may require different series resistance values. The series resistance values should be determined for each application.
2. Although the 82C08 has programmable options, in practice there are only a few choices the designer must make. For iAPX 86/186 systems (CFS = 0) the C2 default mode (pin 33 tied low) is the best choice. This permits zero wait states at 8 and 10 MHz with 150 ns DRAMs. The only consideration is the refresh rate, which must be programmed if the CPU is run at less than 8 MHz.

   For iAPX 286 systems (CFS = 1) the designer must choose between configuration C0 ($\overline{RFS}$ = 0) and C1 ($\overline{RFS}$ = 1, $\overline{FFS}$ = 0). C0 permits zero wait state, 8 MHz iAPX 286 operation with 120 ns DRAMs. However, for consecutive reads, this performance depends on interleaving between two banks. The C1 configuration trades off 1 wait state performance for the ability to use 150 ns DRAMs. 150 ns DRAMs can be supported by the C0 configuration using 7 MHz iAPX 286.
3. For non-Intel microprocessors, the asynchronous command mode would be the best choice, since Intel status lines are not available. To minimize the synchronization delay, the 82C08 should use a 16 MHz clock. The preferred timing configuration is C0.

Table 7. Memory Acknowledge Summary

	Synchronous	Asynchronous	XACK
Fast Cycle	AACK Optimized for Local 80286 (early)	AACK Optimized for Remote 80286 (late)	Multibus Compatible
Slow Cycle	AACK Optimized for Local 8086/186 (early)	AACK Optimized for Remote 8086/186 (late)	Multibus Compatible

POWER DOWN

During Power Down (PD) mode, the 82C08 will perform refresh cycles to preserve the memory content. Two pins are dedicated to this feature, PDD (Power Down Detect) and PDCLK (Power Down Clock). PDD is used to inform the 82C08 of a system power failure, and will remain active as long as the power is down. It is the system's responsibility to detect power failure and to supply this signal. PDCLK is used to supply the clock during power down for the 82C08 refresh circuits. It is the system's responsibility to supply this clock.

Power Supplies

Power down is achieved by eliminating the clock from all the 82C08 circuits that are not participating in the refresh generation. The 82C08 has two power pins (V_{CC}'s), one supplies power to the output buffers and the other, to 82C08 logic. All the active circuits during power down are connected to the logic V_{CC}, including the active output buffers. Therefore, it is the user's choice to connect only the logic V_{CC} pin to the back-up power supply, or to connect both pins to it. It is recommended, however, to connect both pins to the same power supply in order to simplify and to shorten the power up time.

Extended Refresh at Power Down (PD)

To reduce power dissipation during PD, 82C08 will support the extended refresh cycle of the Intel 51CXXL (e.g. 51C64L). In this mode, the refresh period can be extended up to 64 milliseconds versus 4 milliseconds in non-extended cycles. This is achieved by slowing down the PDCLK frequency.

The user should take into consideration that when supporting extended refresh during PD, the dynamic RAM must be refreshed completely within 4 milliseconds, without active cycles, both before going into and after coming out of extended refresh. The 82C08 has the option of performing burst refresh of all the memory whenever the user cannot guarantee the 4 milliseconds idle interval. This is achieved by performing 3 consecutive burst refresh cycles activated internally by the 82C08.

The option of refreshing all the memory is enabled in failsafe mode configuration (RFRQ input high at reset). When 82C08 detects power down, (high level at PDD) it examines the RFRQ input. High level at the RFRQ input will cause 3 burst refresh cycles to be performed. The user should supply the power and the system clock during the time interval of the 3 burst cycles, e.g. 2310 clock cycles after activating PDD. Low level at RFRQ input enables the 82C08 to enter power down immediately without executing any bursts.

Power Down Procedure

The 82C08 will preserve the memory content during the entire period of the system operation. Upon detection of power down, the 82C08 will save internally its configuration status and the refresh address counter content, execute 3 burst refresh cycles. (If it is programmed to failsafe mode and the RFRQ input level is high), it will switch the internal clock from the system clock (CLK) to the power down clock (PDCLK) and will continue the refresh to the next address location. (See Figure 11.)

When power is up again (PDD input deactivated), the 82C08 will issue internal reset which will not reprogram the device and will not clear the refresh address counter, and therefore, refresh will continue to the next address location. After the internal reset, 82C08 performs 3 burst refresh cycles which refresh the whole memory, as at entering extended PD. This is done to give the 82C08 enough time to wake up. Notice, at the time interval of 2310 clocks after power recovering no memory access will be performed.

82C08 Outputs on Power Down

Four of the 82C08 outputs are not activated during power down, $\overline{AACK}$, $\overline{CAS}$0–1 and WE. All these outputs will be forced to a non-active state, $\overline{AACK}$ and $\overline{CAS}$0–1 will be forced high and WE will be forced low (External NAND buffer is used to drive the $\overline{WE}$ DRAM inputs, hence a high level on the DRAM inputs). The other 82C08 outputs, AO0–9 and $\overline{RAS}$0–1, will switch to perform the memory refresh in a "$\overline{RAS}$-ONLY REFRESH CYCLE." The $\overline{RAS}$ outputs internal pull-ups assure high levels on these outputs, as close as possible to V_{CC}, for low DRAM power. The size of the output buffers, in power down, is smaller than the normal size, and therefore, the speed of these buffers is slower. It is done in order to reduce the speed of charging and discharging the outputs and hence reduce spikes on the power lines. It is required especially in power down, since there is only one power supply pin active which drives the output buffers as well as the internal logic.

All the device inputs, beside PDD and PDCLK, will be ignored during power down.

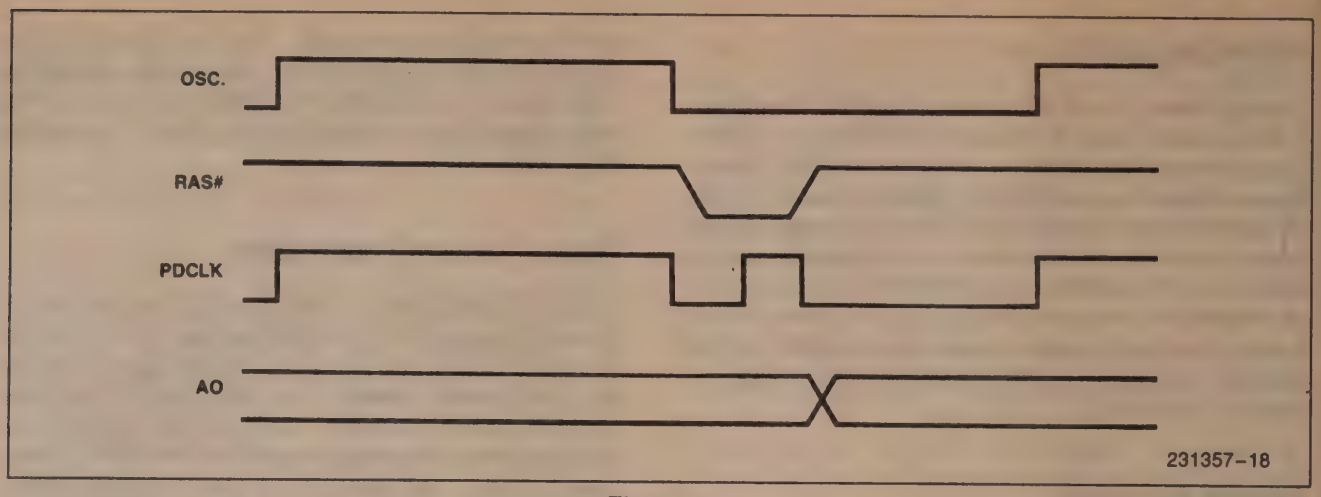


Figure 8

Power Down Detect

As previously mentioned, the PDD input will be supplied by the system to inform the 82C08 of a power failure. It can be asynchronous since the 82C08 synchronizes it internally. The PDD input will be sampled by the 82C08 before the beginning of every memory cycle but only after the termination of programming and initialization period. The user should guarantee V_{CC} and CLK stable during the programming and initialization period (300 clocks after RESET). If the whole memory refresh is required (for extended refresh) then V_{CC} and system clock should be available 2310 clocks after activating PDD. If it isn't required then 82C08 should wait for present memory cycle completion and synchronization time which will take about 25 system clock cycles.

With PDD going inactive, the 82C08 synchronizes the clock back to the CLK clock, issuing internal reset and will perform 3 burst refresh cycles.

NOTE:
The power supplies and the CLK should go up before the PDD is deactivated. All CPU requests will be ignored when PDD is active.

Refresh during Power Down

The 82C08 has two clock pins, CLK is the system clock and PDCLK is the power down clock. PDCLK should be an independent clock which has its own crystal oscillator. When entering power down, the 82C08 will disable the system clock internally and will run with the PDCLK. The system clock will be enabled and the PDCLK will be disabled when power is up. The CLK and PDCLK will be switched internally for the refresh circuits.

During power down, '$\overline{RAS}$-ONLY REFRESH' will be performed by the 82C08. The time interval between refreshes is 5 PDCLKs and this is fixed for all applications. However, the 82C08 can support the extended refresh (up to 64 ms) by slowing down the PDCLK frequency.

During the power down refresh cycle, $\overline{RAS}$ will be activated for one PDCLK cycle only. In extended refresh, the PDCLK frequency will be below 50 kHz and this will cause a long duration of the $\overline{RAS}$ signal which will increase the DRAM's current rapidly. To minimize the $\overline{RAS}$ low pulse, the two RC networks shown in Figure 9 are designed to insert one very fast (1 μs) cycle whenever $\overline{RAS}$ is low (see Figure 8). The time constant of RC1 and RC2 should be centered around 300 ns and 100 ns respectively.

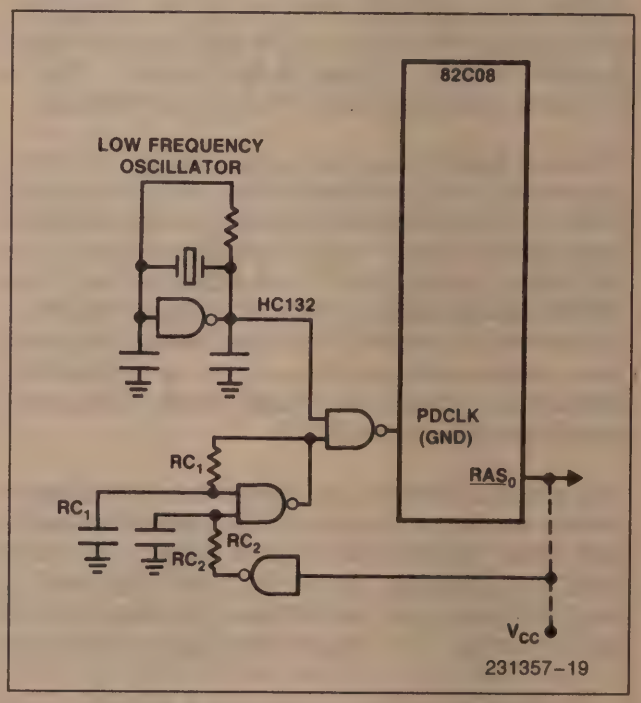


Figure 9. Low Frequency Oscillator

Power Down Synchronization

The 82C08 main clock (MCLK) is generated internally, from the system clock (CLK) and the power down clock (PDCLK) (see Figure 10), and is driving the circuits that are active at all times, i.e.: circuits that are active both in power down mode and in normal operation. The system clock (CLK) is driving the circuits that are active in normal operation only, and the PDCLK is driving the circuits that are active in power down only. The operation of the three clocks is as follows:

When entering power down mode, and the whole memory refresh is required, the CLK minimum active time after PDD is activated is 2310 clocks.

When it isn't required, PDCLK should be active, and CLK should remain active for at least 22 clock cycles + synchronization time, for completion of SBE write. The synchronization time is the ratio of PDCLK and CLK + 1. Therefore, the CLK minimum active time after PD is activated:

22 + [CLK(MHz) / PDCLK(MHz) + 1] clock cycles

When the power is up again, PDCLK should remain active at least 4 clock cycles after PD is going inactive, to assure completion of refresh cycle and internal synchronization time.

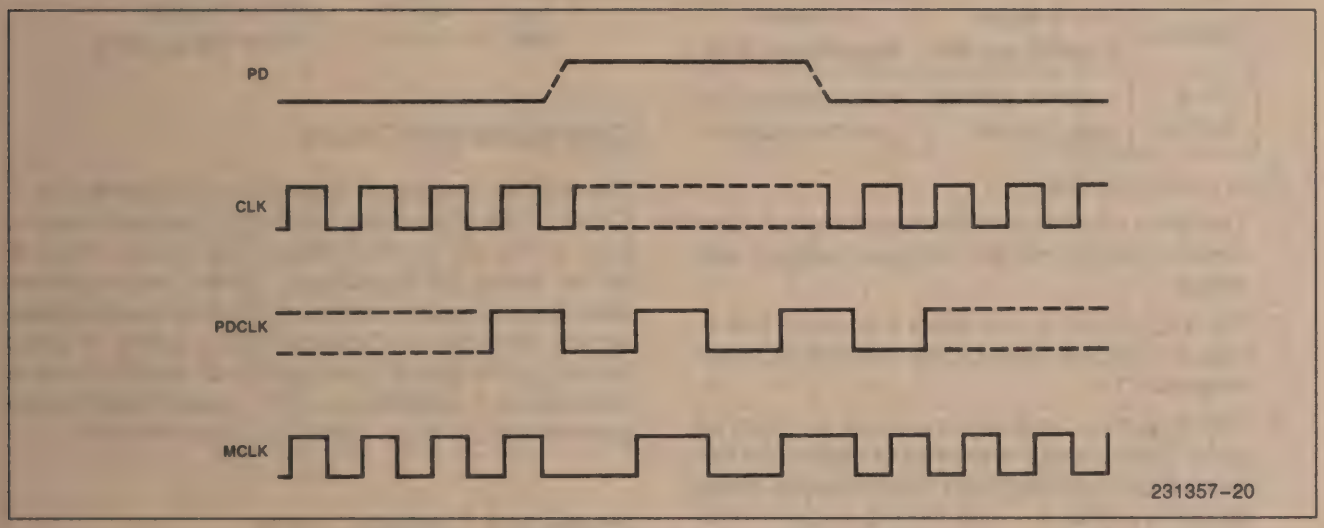


Figure 10

POWER DOWN FLOW

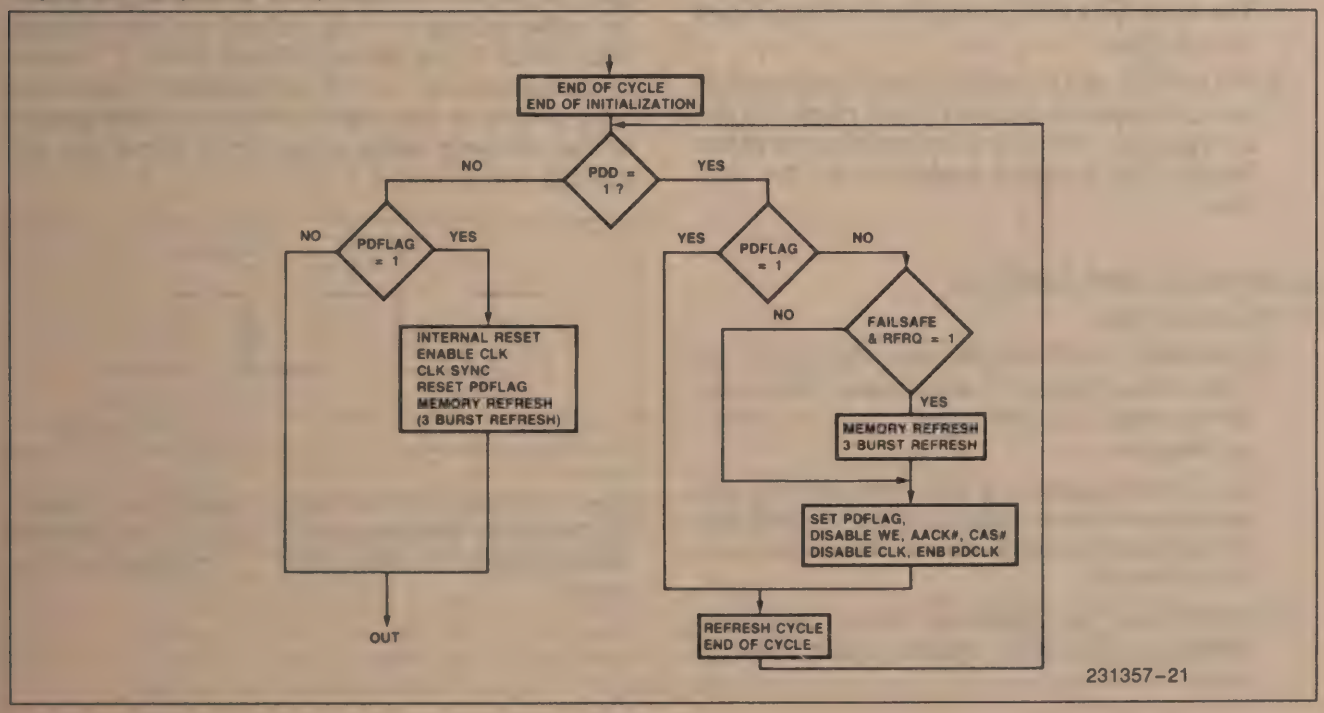


Figure 11

Differences Between 8208 and 82C08

The differences between the HMOS 8208 and the CHMOS 82C08 represent forward compatible enhancements. The 82C08 can be plugged into an 8208 socket without changes.

LOGICAL DIFFERENCES

1. 82C08 has one new feature:

   Power Down (PD)

2. 82C08 supports CMOS DRAMs with T_{RAC} 100, 150

3. Address Mapping:

Outputs	9 Most Significant Bits	9 Least Significant Bits
8208	column address	row address
82C08	row address	column address

4. Slow cycle shortening:

   1). The write cycle is two clocks shorter so consecutive writes will be executed without wait states.

   2) The WE output is two clocks shorter. Therefore, an external latch on the WE output is not necessary.

   3) CAS output is shorter by one clock on the read cycle. This reduces one level of buffers for address/data bus needed in 8208 designs. Read access margins are improved to support non-Intel spec. RAMs.

5. Fast cycle shortening:

   1) The write cycle in C0 configuration is shortened by one clock.

   2) For both C0 and C1 synchronous configuration, the CAS signal is shorter by one clock and the activation of RAS is tied to the 02 cycle of the 80286. This prevents contention on the data bus.

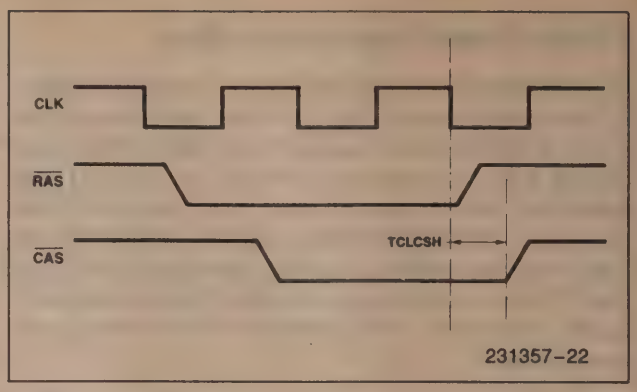
ELECTRICAL DIFFERENCES

1. AC parameters:

   1) CAS delay: In C2 synchronous read cycle, the CAS is deactivated by some delay from clock falling edge (TCLCSH timing) as in the following diagram:

   In C2 write cycles the $\overline{CAS}$ activation is triggered by the clock falling edge with a delay of 35 ns from the clock. For 8208 the delay is TP/1.8 + 53.

   2) 82C08 has an additional timing parameter TKNVCH ($\overline{RD}$, $\overline{WR}$) inactive setup time to clock.



2. DC parameters: The difference is in the current consumption.

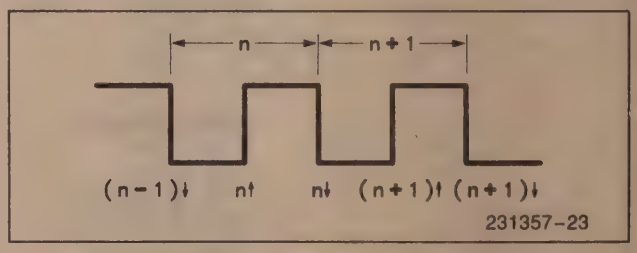
	8208	82C08
I_{CC}	300 mA	30 mA (typical)
IPD	—	1 mA* (estimated)

Configuration Charts

The 82C08 operates in three basic configurations—C0, C1, C2—depending upon the programming of CFS (PD0), $\overline{RFS}$ (PD2), and $\overline{FFS}$ (PD7). Table 8 shows these configurations. These modes determine the clock edges for the 82C08's programmable signals, as shown in Table 9. Finally, Table 10 gives the programmable AC parameters of the 82C08 as a function of configuration. The non-programmable parameters are listed under AC Characteristics.

Using the Timing Charts

The notation used to indicate which clock edge triggers an output transition is "n↑" or "n↓", where "n" is the number of clock periods that have passed since clock 0, the reference clock, and "↑" refers to rising edge and "↓" to falling edge. A clock period is defined as the interval from a clock falling edge to the following falling edge. Clock edges are defined as shown below.



The clock edges which trigger transitions on each 82C08 output are tabulated in Table 9. "H" refers to the high-going transition, and "L" to low-going transition.

Clock 0 is defined as the clock in which the 82C08 begins a memory cycle, either as a result of a port request which has just arrived, or of a port request which was stored previously but could not be serviced at the time of its arrival because the 82C08 was performing another memory cycle. Clock 0 is identified externally by the leading edge of RAS, which is always triggered on 0 ↓.

Table 8. 82C08 Configurations

Timing Conf.	CFS(PD0)	RFS(PD2)	FFS(PD7)	Wait States*
C_0	iAPX286(1)	FAST RAM(1)	20 MHz(1)	0
C_0	iAPX286(1)	FAST RAM(1)	16 MHz(1)	0
C_1	iAPX286(1)	SLOW RAM(0)	16 MHz(1)	1
C_0	iAPX286(1)	FAST RAM(1)	10 MHz (0)	0
C_0	iAPX286(1)	SLOW RAM(0)	10 MHz (0)	0
C_2	iAPX186(0)	DON'T CARE	DON'T CARE	0

* Using $\overline{EAACK}$ (synchronous mode)

Table 9a. Timing Chart — Synchronous Mode

		$\overline{RAS}$		ADDRESS		$\overline{CAS}$		WE		$\overline{EAACK}$	
Cn	Cycle	L	H	Col	Row*	L	H	H	L	L	H
0	RD,RF	0↓	3↓	0↓	2↓	1↓	3↓			1↓	4↓
	WR	0↓	4↓	0↓	3↓	2↓	4↓	1↓	4↓	1↓	4↓
1	RD,RF	0↓	4↓	0↓	3↓	1↓	5↓			2↓	5↓
	WR	0↓	5↓	0↓	3↓	2↓	5↓	1↓	5↓	2↓	5↓
2	RD,RF	0↓	2↓	0↓	2↓	0↓	2↓			0↓	2↓
	WR	0↓	2↓	0↓	3↓	1↓	3↓	0↓	2↓	0↓	2↓

Table 9b. Timing Chart — Asynchronous Mode

		$\overline{RAS}$		ADDRESS		$\overline{CAS}$		WE		$\overline{LAACK}$		$\overline{XAACK}$	
Cn	Cycle	L	H	Col	Row*	L	H	H	L	L	H	L	H
0	RD,RF	0↓	3↓	0↓	2↓	1↓	4↓			2↓	5↓	3↓	$\overline{RD}$
	WR	0↓	4↓	0↓	3↓	2↓	4↓	1↓	4↓	1↓	4↓	3↓	$\overline{WR}$
1	RD,RF	0↓	4↓	0↓	3↓	1↓	6↓			2↓	5↓	4↓	$\overline{RD}$
	WR	0↓	5↓	0↓	3↓	2↓	5↓	1↓	5↓	1↓	4↓	3↓	$\overline{WR}$
2	RD,RF	0↓	2↓	0↓	2↓	0↓	3↓			1↓	3↓	2↓	$\overline{RD}$
	WR	0↓	2↓	0↓	3↓	1↓	3↓	0↓	2↓	1↑	3↑	2↓	$\overline{WR}$

The only difference between the two tables is the trailing edge of CAS for all read cycle configurations. In asynchronous mode, CAS trailing edge is one clock later than in synchronous mode.

NOTES FOR INTERPRETING THE TIMING CHART:

1. COLUMN ADDRESS is the time column address becomes valid.
2. The $\overline{CAS}$, $\overline{EAACK}$, $\overline{LAACK}$ and $\overline{XACK}$ outputs are not issued during refresh.
3. $\overline{XACK}$—high is reset asynchronously by command going inactive and not by a clock edge.
4. $\overline{EAACK}$ is used in synchronous mode, $\overline{LAACK}$ and $\overline{XACK}$ in asynchronous mode.
5. ADDRESS-Row is the clock edge where the 82C08 A0 switches from current column address to the next row address.
6. If a cycle is inhibited by PCTL = 1 (Multibus I/F mode) then $\overline{CAS}$ is not activated during write cycle and $\overline{XACK}$ is not activated in either read or write cycles.

*Column addresses switch to row addresses for next memory cycle. The row address buffer is transparent following this clock edge. 'TRAH' specification is guaranteed as per data sheet.

82C08—DRAM Interface Parameter Equations

Several DRAM parameters, but not all, are a direct function of 82C08 timings, and the equations for these parameters are given in the following tables. The following is a list of those DRAM parameters which have NOT been included in the following tables, with an explanation for their exclusion.

WRITE CYCLE

tDS: system-dependent parameter.
tDH: system-dependent parameter.
tDHR: system-dependent parameter.

READ, WRITE REFRESH CYCLES

tRAC: response parameter.
tCAC: response parameter.
tREF: See "Refresh Period Options".
tCRP: must be met only if CAS-only cycles, which do not occur with 82C08, exist.
tRAH: See "A.C. Characteristics"
tRCD: See "A.C. Characteristics"
tASC: See "A.C. Characteristics"
tASR: See "A.C. Characteristics"
tOFF: response parameter.

Table 10. Programmable Timings

Read and Refresh Cycles

Parameter	C2-Slow Cycle	C0-Fast Cycle	C1-Fast Cycle	Notes
tRP	2TCLCL-T25	3TCLCL-T25	3TCLCL-T25	1
tCPN	1.5TCLCL-T34	3TCLCL-T34	2TCLCL-T34	1, 5
tCPN	2.5TCLCL-T34	4TCLCL-T34	3TCLCL-T34	1, 4
tRSH	2TCLCL-T32	2TCLCL-T33	3TCLCL-T33	1
tCSH	3TCLCL-T25	4TCLCL-T25	6TCLCL-T25	1, 5
tCSH	2TCLCL + T34(min)-T25	3TCLCL-T25	5TCLCL-T25	1, 4
tCAH	2TCLCL-T32	TCLCL-T33	2TCLCL-T33	1
tAR	2TCLCL-T25	2TCLCL-T25	3TCLCL-T25	1
tT	3/30	3/30	3/30	2
tRC	4TCLCL	6TCLCL	7TCLCL	1
tRAS	2TCLCL-T25	3TCLCL-T25	4TCLCL-T25	1
tCAS	3TCLCL-T32	3TCLCL-T33	5TCLCL-T33	1, 5
tCAS	2TCLCL + T34(min)-T32	2TCLCL-T25	4TCLCL-T33	1, 4
tRCS	1.5TCLCL-TCL-T36-TBUF	2TCLCL-TCL-T36-TBUF	2TCLCL-TCL-T36-TBUF	1
tRCH	TCLCL-T32 + T36 (min.)	TCLCL-T32	TCLCL-T32	1

Write Cycles

Parameter	C2-Slow Cycle	C0-Fast Cycle	C1-Fast Cycle	Notes
tRP	2TCLCL-T25	3TCLCL-T25	3TCLCL-T25	1
tCPN	1.5TCLCL-T34	5TCLCL-T34	4TCLCL-T34	1
tRSH	3TCLCL-T32	2TCLCL-T33	3TCLCL-T33	1
tCSH	3TCLCL-T25	4TCLCL-T25	5TCLCL-T25	1
tCAH	2TCLCL-T32	TCLCL-T33	TCLCL-T33	1
tAR	3TCLCL-T25	3TCLCL-T25	3TCLCL-T25	1
tT	3/30	3/30	3/30	2
tRC	4TCLCL	7TCLCL	8TCLCL	1
tRAS	2TCLCL-T25	4TCLCL-T25	5TCLCL-T25	1
tCAS	2TCLCL-T32	2TCLCL-T33	3TCLCL-T33	1
tWCH	TCLCL-T32	2TCLCL-T33	3TCLCL-T33	1,3
tWCR	2TCLCL-T25	4TCLCL-T25	5TCLCL-T25	1,3
tWP	2TCLCL-T36-TBUF	3TCLCL-T36-TBUF	4TCLCL-T36-TBUF	1
tRWL	2TCLCL-T36-TBUF	3TCLCL-T36-TBUF	4TCLCL-T36-TBUF	1
tCWL	3TCLCL-T36-TBUF	3TCLCL-T36-TBUF	4TCLCL-T36-TBUF	1
tWCS	TCLCL + T36-T31-TBUF	TCLCL + T36-T31-TBUF	TCLCL + T36-T31-TBUF	1

NOTES:
1. Minimum.
2. Value on right is maximum; value on left is minimum.
3. Applies to the eight warm-up cycles during initialization.
4. For synchronous mode only.
5. For asynchronous mode only.

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias	−0°C to +70°C
Storage Temperature	−65°C to +150°C
Voltage On Any Pin With Respect to Ground	−0.5V to +7V
Power Dissipation	0.5 Watts

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

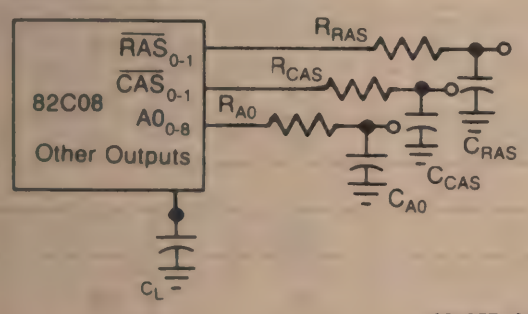
NOTICE: Specifications contained within the following tables are subject to change.

D.C. CHARACTERISTICS T_A = 0°C to 70°C; V_{CC} = 5.0V ±10%; V_{SS} = GND

Symbol	Parameter	Min	Max	Units	Comments
V_{IL}	Input Low Voltage	−0.5	+0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC} + 0.5	V	
V_{OL}	Output Low Voltage		0.45	V	Note 1
V_{OH}	Output High Voltage	2.4		V	Note 1
V_{ROL}	RAM Output Low Voltage		0.45	V	Note 1
V_{ROH}	RAM Output High Voltage	2.6		V	Note 1
I_{CC}	Supply Current		30 60	mA mA	Note 3 T_A = 0°
I_{LI}	Input Leakage Current		±10	μA	$0V \le V_{IN} \le V_{CC}$
V_{CL}	Clock Input Low Voltage	−0.5	+0.6	V	
V_{CH}	Clock Input High Voltage	3.8	V_{CC} + 0.5	V	
C_{IN}	Input Capacitance		20	pF	fc = 1 MHz
V_{OHPD}	RAS Output High Power Down	V_{CC} − 0.5		V	Note 2
I_{PD}	Supply Current at Power Down		1.0	mA	Estimated Value

NOTE:
1. I_{OL} = 5 mA and I_{OH} = −0.32 mA WE: I_{OL} = 8 mA
2. $\overline{RAS}$ Output voltage during power down.
3. Typical value.

A.C. Testing Load Circuit

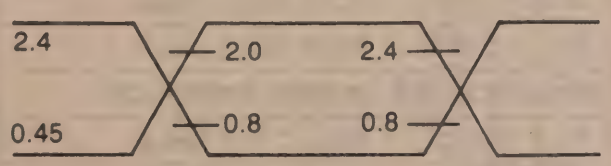


231357-24

R_{RAS} = 39Ω
R_{CAS} = 39Ω
R_{A0} = 22Ω

C_{RAS} = 150 pF
C_{CAS} = 150 pF
C_{A0} = 200 pF
C_L = 50 pF

A.C. Testing Input, Output Waveform



231357-25

A.C. Testing inputs (except clock) are driven at 2.4V for a logic "1" and 0.45V for a logic "0" (clock is driven at 4.0V and 0.45V for logic "1" and "0" respectively). Timing measurements are made at 2.0V, 2.4V for logic "1" and 0.8V for logic "0".

A.C. CHARACTERISTICS (T_A = 0°C to 70°C; V_{CC} = +5V ±10%, V_{SS} = 0V)

Measurements made with respect to RAS_{0-1}, CAS_{0-1}, AO_{0-8}, are at +2.4V and 0.8V CLK at 3V, 1V. All other pins are measured at 2.0V and 0.8V. All times are ns unless otherwise indicated. Testing done with specified test load.

Ref	Symbol	Parameter	82C08-20, 82C08-16 82C08-10, 82C08-8		82C08-12		Units	Notes
			Min	Max	Min	Max		
CLOCK AND PROGRAMMING								
	tF	Clock Fall Time		12		12	ns	3
	tR	Clock Rise Time		12		12	ns	3
1	TCLCL	Clock Period						
		82C08-20	50	250			ns	1
		82C08-16	62.5	250			ns	1
		82C08-12			83.3	250	ns	1
		82C08-10	100	500			ns	2
		82C08-8	125	500			ns	2
2	TCL	Clock Low Time						
		82C08-20	10	230			ns	1
		82C08-16	15	230			ns	1
		82C08-12			20	225	ns	1
		82C08-10	44				ns	2
		82C08-8	TCLCL/2-12				ns	2
3	TCH	Clock High Time						
		82C08-20	17	230			ns	1
		82C08-16	20	230			ns	1
		82C08-12			25	230	ns	1
		82C08-10	44				ns	2
		82C08-8	TCLCL/3+2				ns	2
4	TRTVCL	Reset to CLK ↓ Setup	40		65		ns	4
5	TRTH	Reset Pulse Width	4TCLCL		4TCLCL		ns	
6	TPGVRTL	PCTL, PDI, RFRQ to RESET ↓ Setup	125		167		ns	5
7	TRTLPGX	PCTL, RFRQ to RESET ↓ Hold	10		10		ns	
8	TCLPC	PCLK from CLK ↓ Delay		45		55	ns	
9	TPDVCL	PDI to CLK ↓ Setup	60		85		ns	
10	TCLPDX	PDI to CLK ↓ Hold	40		55		ns	6
SYNCHRONOUS μP PORT INTERFACE								
11	TPEVCL	PE to CLK ↓ Setup	30		40			2
12	TKVCL	$\overline{RD}$, $\overline{WR}$, $\overline{PE}$, PCTL to CLK ↓ Setup	20		25		ns	1
13	TCLKX	$\overline{RD}$, $\overline{WR}$, $\overline{PE}$, PCTL to CLK ↓ Hold	0		0		ns	
14	TKVCH	RD, WR, PCTL to CLK ↑ Setup	20		30		ns	2

A.C. CHARACTERISTICS (Continued)

Ref	Symbol	Parameter	82C08-20, 82C08-16 82C08-10, 82C08-8		82C08-12		Units	Notes
			Min	Max	Min	Max		
ASYNCHRONOUS μP PORT INTERFACE								
15	TRWVCL	$\overline{RD}$, $\overline{WR}$ to CLK ↓ Setup	20		30		ns	8.9
16	TRWL	$\overline{RD}$, $\overline{WR}$ Pulse Width	2TCLCL+30		2TCLCL+40		ns	
17	TRWLPEV	$\overline{PE}$ from $\overline{RD}$, $\overline{WR}$ ↓ Delay CFS=1 CFS=0		 TCLCL-20 TCLCL-30		 TCLCL-30	 ns ns	 1 2
18	TRWLPEX	$\overline{PE}$ to $\overline{RD}$, $\overline{WR}$ ↓ Hold	2TCLCL+30		2TCLCL+40		ns	
19	TRWLPTV	PCTL from $\overline{RD}$, $\overline{WR}$ ↓ Delay		TCLCL-30		TCLCL-40	ns	2
20	TRWLPTX	PCTL to $\overline{RD}$, $\overline{WR}$ ↓ Hold	2TCLCL+30		2TCLCL+40		ns	2
21	TRWLPTV	PCTL from $\overline{RD}$, $\overline{WR}$ ↓ Delay		2TCLCL-20		2TCLCL-30	ns	1
22	TRWLPTX	PCTL to $\overline{RD}$, $\overline{WR}$ ↓ Hold	3TCLCL+30		3TCLCL+40		ns	1
RAM INTERFACE								
23	TAVCL	AL, AH, BS to CLK ↓ Set-up 82C08-20 82C08-16	45+tASR 51+tASR 45+tASR		55+tASR		ns ns ns	2
24	TCLAX	AL, AH, BS to CLK ↓ Hold	0		0		ns	
25	TCLRSL	RAS ↓ from CLK ↓ Delay		25 35 60		35 60	ns ns ns	1 2 24
26	TRCD	$\overline{RAS}$ to $\overline{CAS}$ Delay CFS=1 CFS=0 CFS=0 CFS=0	 TCLCL-25 30 TCLCL/2-30 60		 TCLCL-30		 ns ns ns ns	 1, 14 23 2, 11, 14 2, 12, 14
27	TCLRSH	$\overline{RAS}$ ↑ from CLK ↓ Delay		50 60		60 60	ns ns	 24

A.C. CHARACTERISTICS (Continued)

Ref	Symbol	Parameter	82C08-20, 82C08-16 8208-10, 82C08-8		82C08-12		Units	Notes
			Min	Max	Min	Max		
RAM INTERFACE (Continued)								
28	TRAH	82C08-20	18					13, 15
		82C08-16	TCLCL/2-13		TCLCL/2-15		ns	13, 15
		CFS=0	TCLCL/4-10				ns	2, 11, 15
		CFS=0	20				ns	23
29	TASR	Row A0 $\overline{RAS}$ ↓ Setup						10, 16
30	TASC	Column A0 to $\overline{CAS}$ ↓ Setup						
		CFS=1	2		5		ns	1, 13, 17, 18
		CFS=0	5				ns	2, 13, 17, 18
		CFS=0	5				ns	23
31	TCAH	Column A0 to $\overline{CAS}$ Hold	(See DRAM Interface Tables)					
32	TCLCSL	$\overline{CAS}$ ↓ from CLK ↓ Delay						
		CFS=0	TCLCL/4+30	TCLCL/1.8+53			ns	2, 26
		CFS=0	50	100			ns	23
		CFS=0		35		40	ns	2, 27
		CFS=1		35		40	ns	1
34	TCLCSH	$\overline{CAS}$ ↑ from CLK ↓ Delay	TCLCL/4	50		60	ns	22
				$\frac{TCLCL}{3.2}+50$			ns	
35	TCLWL	WE ↓ from CLK ↓ Delay		35		45	ns	
36	TCLWH	WE ↑ from CLK ↓ Delay						
		CFS=0	TCLCL/4+30	TCLCL/1.8+53				2
		CFS=1		35		45	ns	1
		CFS=0	50	100			ns	23
37	TCLTKL	$\overline{XACK}$ ↓ from CLK ↓ Delay		35		45	ns	
38	TRWLTKH	$\overline{XACK}$ ↑ from $\overline{RD}$ ↑, $\overline{WR}$ ↑ Delay		50		55	ns	
39	TCLAKL	$\overline{AACK}$ ↓ from $\overline{CLK}$ ↓ Delay		35		35	ns	
40	TCLAKH	$\overline{AACK}$ ↑ from CLK ↓ Delay		50		60	ns	

A.C. CHARACTERISTICS (Continued)

Ref	Symbol	Parameter	82C08-20, 82C08-16 82C08-10, 82C08-8		82C08-12		Units	Notes
			Min	Max	Min	Max		
REFRESH REQUEST								
41	TRFVCL	RFRQ to CLK ↓ Setup	20		30		ns	
42	TCLRFX	RFRQ to CLK ↓ Hold	10		10		ns	
43	TFRFH	Failsafe RFRQ Pulse Width	TCLCL + 30		TCLCL + 50		ns	19
44	TRFXCL	Single RFRQ Inactive to CLK ↓ Setup	20		30		ns	20
45	TBRFH	Burst RFRQ Pulse Width	2TCLCL + 30		2TCLCL + 50		ns	19
46	TPDVCL	PDD Setup Time	20		30		ns	24, 25
47	TPDHRFX	RFRQ Valid after PDD Active	4TCLCL + 20		4TCLCL + 30			
48	RFVPDH	RFRQ Setup Time to PDD Active	0		0			24

The following RC loading is assumed:

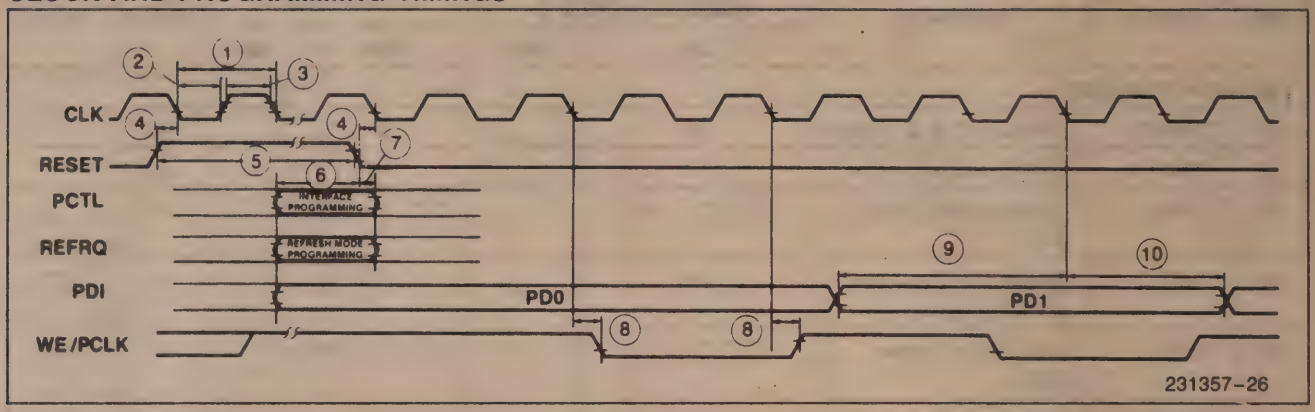
$A0_{0-8}$	R = 22Ω	C = 200 pF
RAS_{0-1}, CAS_{0-1}	R = 39Ω	C = 150 pF
AACK, WE/PCLK	R = 33Ω	C = 50 pF

NOTES:

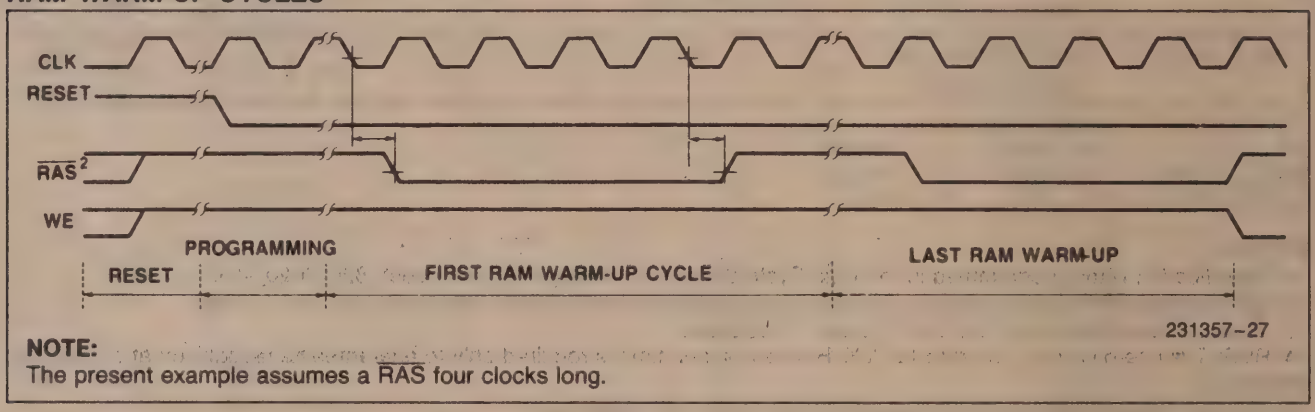
1. Specification when programmed in the Fast Cycle processor mode (iAPX 286 mode). 82C08-20, -16, -12 only.
2. Specification when programmed in the Slow Cycle processor mode (iAPX 186 mode). 82C08-8, -6 only.
3. tR and tF are referenced from the 3.5V and 1.0V levels.
4. RESET is internally synchronized to CLK. Hence a set-up time is required only to guarantee its recognition at a particular clock edge.
5. The first programming bit (PD0) is also sampled by RESET going low.
6. TCLPDX is guaranteed if programming data is shifted using PCLK.
8. TRWVCL is not required for an asynchronous command except to guarantee its recognition at a particular clock edge.
9. Valid when programmed in either Fast or Slow Cycle mode.
10. tASR is a user specified parameter and its value should be added accordingly to TAVCL.
11. When programmed in Slow Cycle mode and 125 ns ≤ TCLCL < 200 ns.
12. When programmed in Slow Cycle mode and 200 ns ≤ TCLCL.
13. Specification for Test Load conditions.
14. tRCD (actual) = tRCD (specification) $+0.06\ (\Delta C_{RAS}) - 0.06(\Delta C_{CAS})$ where ΔC = C (test load) − C (actual) in pF. (These are first order approximations.)
15. tRAH (actual) = tRAH (specification) $+ 0.06\ (\Delta C_{RAS}) - 0.022\ (\Delta C_{A0})$ where ΔC = C (test load) − C (actual) in pF. (These are first order approximations.)
16. tASR (actual) = tASR (specification) $+0.06\ (\Delta C_{A0}) - 0.025\ (\Delta C_{RAS})$ where ΔC = C (test load) − C (actual) in pF. (These are first order approximations.)
17. tASC (actual) = tASC (specification) $+0.06\ (\Delta C_{A0}) - 0.025\ (\Delta C_{CAS})$ where ΔC (test load) − C (actual) in pF. (These are first order approximations.)
18. tASC is a function of clock frequency and thus varies with changes in frequency. A minimum value is specified.
19. TFRFH and TBRFH pertain to asynchronous operation only.
20. Single RFRQ should be supplied synchronously to avoid burst refresh.
22. CFS = 0, synchronous mode, Read cycle.
23. For 10 MHz Slow Cycle only.
24. Power down mode.
25. PDD is internally synchronized. A setup time is required only to guarantee its recognition at a particular clock edge.
26. Slow Cycle Read only.
27. Slow Cycle Write only.

WAVEFORMS

CLOCK AND PROGRAMMING TIMINGS

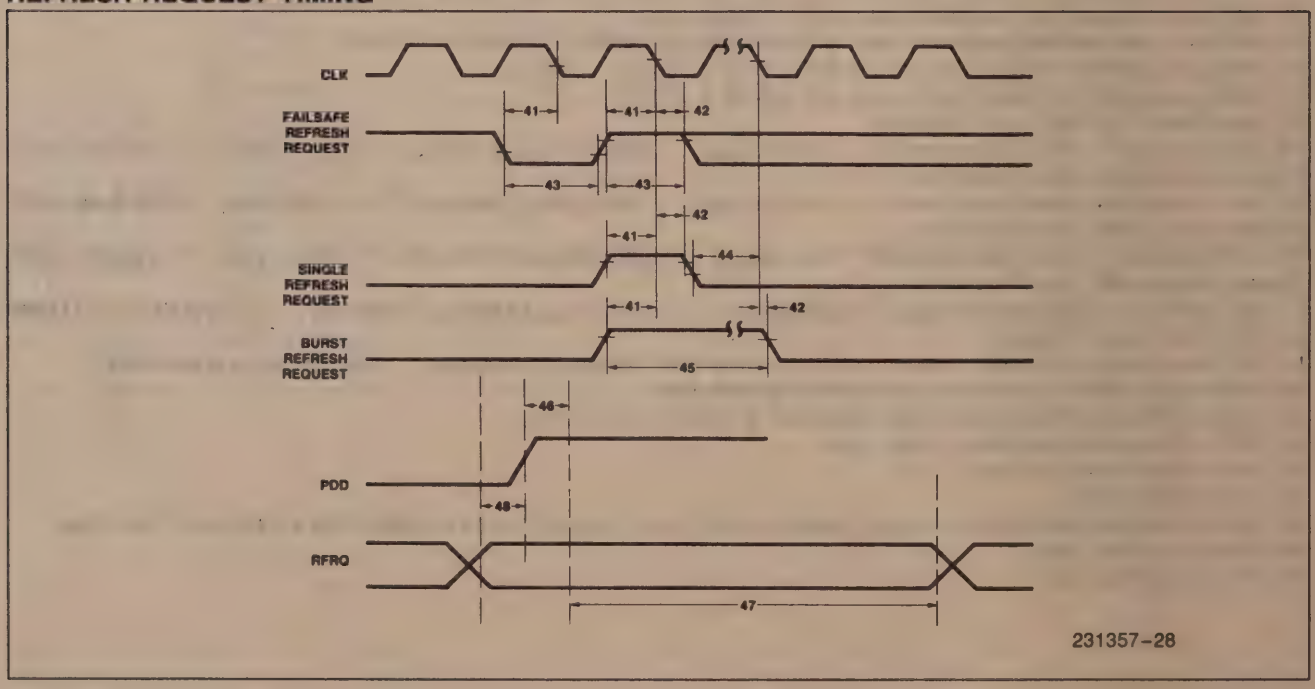


RAM WARM-UP CYCLES



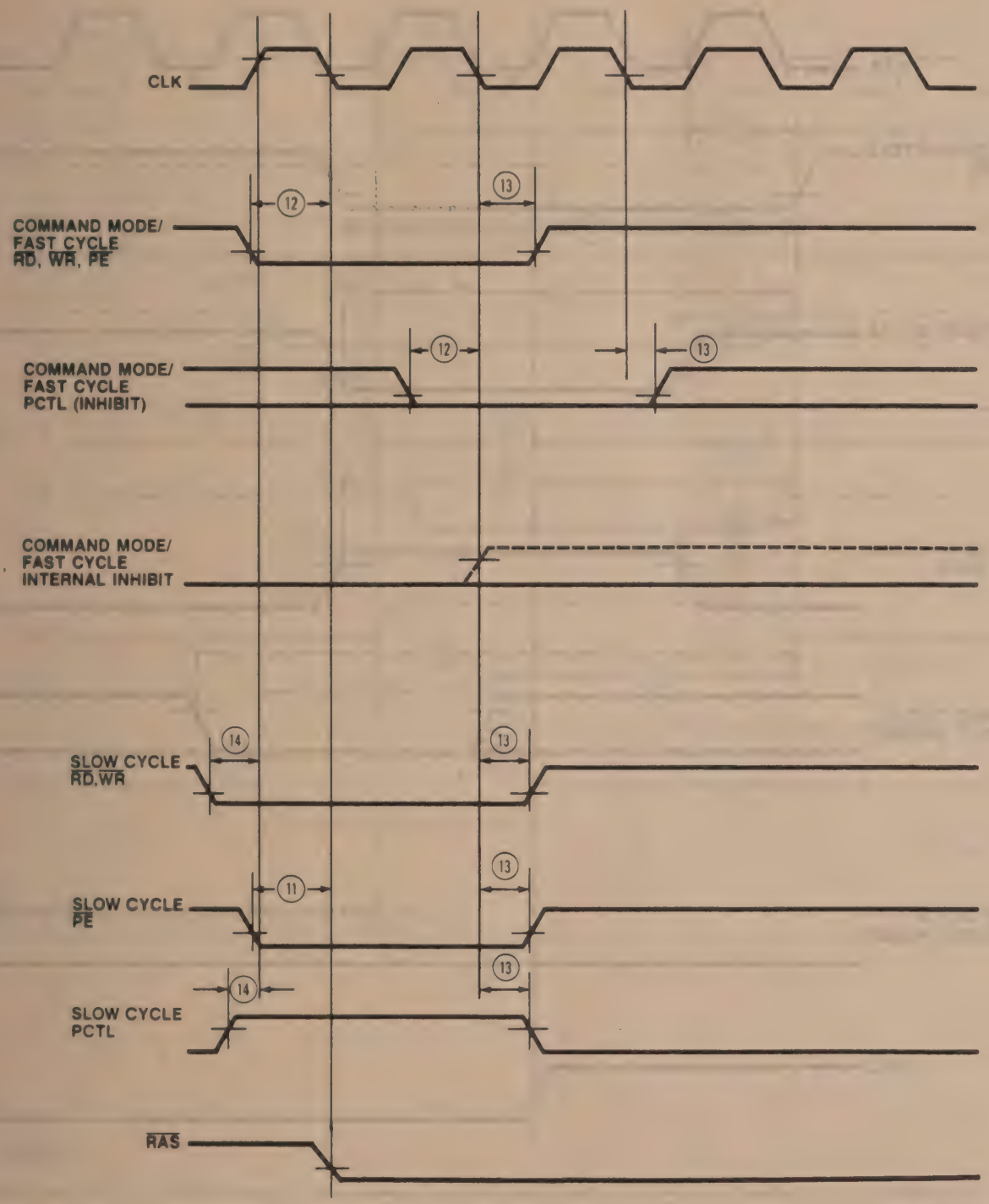
NOTE:
The present example assumes a $\overline{RAS}$ four clocks long.

REFRESH REQUEST TIMING



WAVEFORMS (Continued)

SYNCHRONOUS PORT INTERFACE

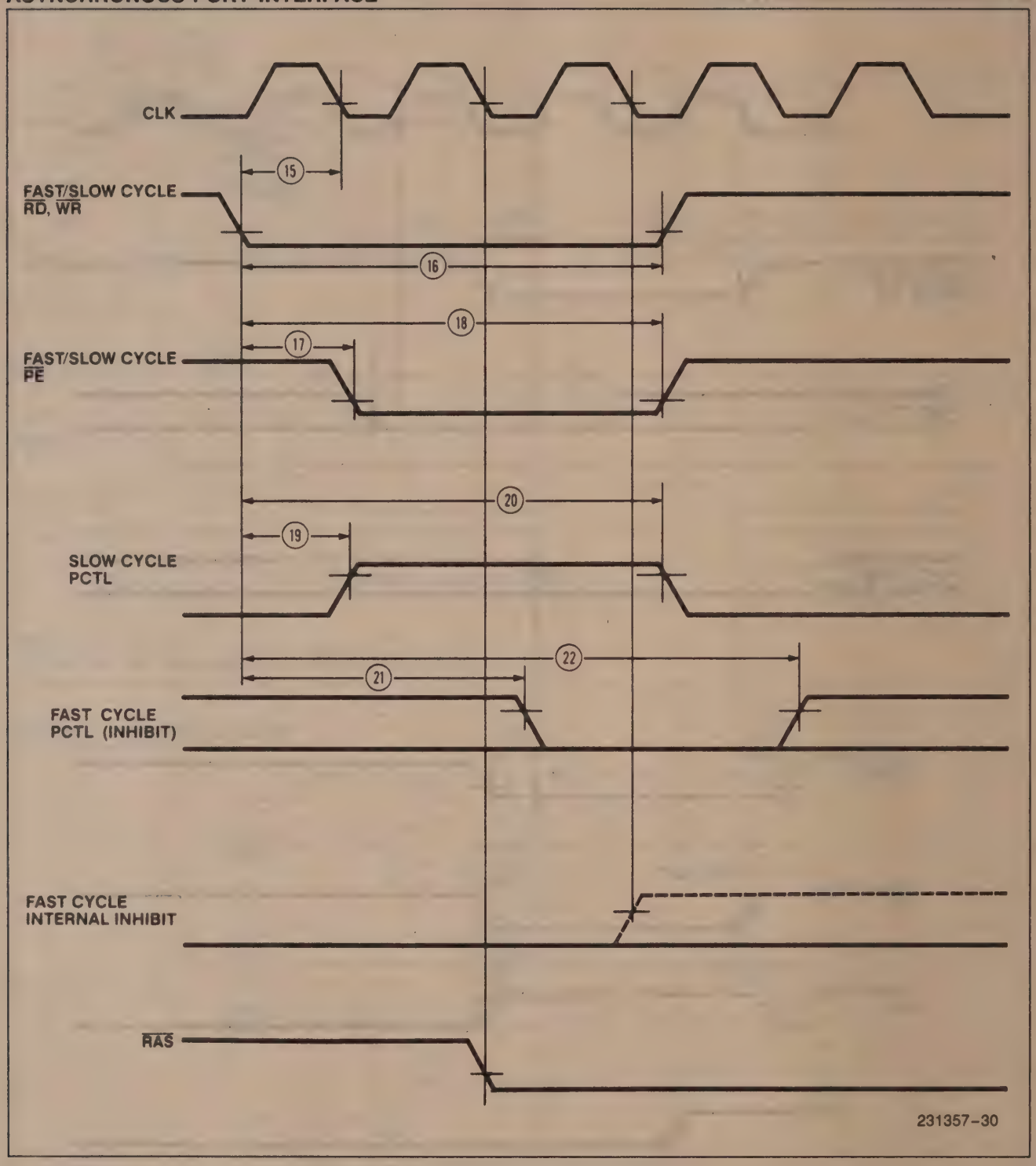


231357-29

NOTE:
Actual transitions are programmable. Refer to Tables 8 and 9.

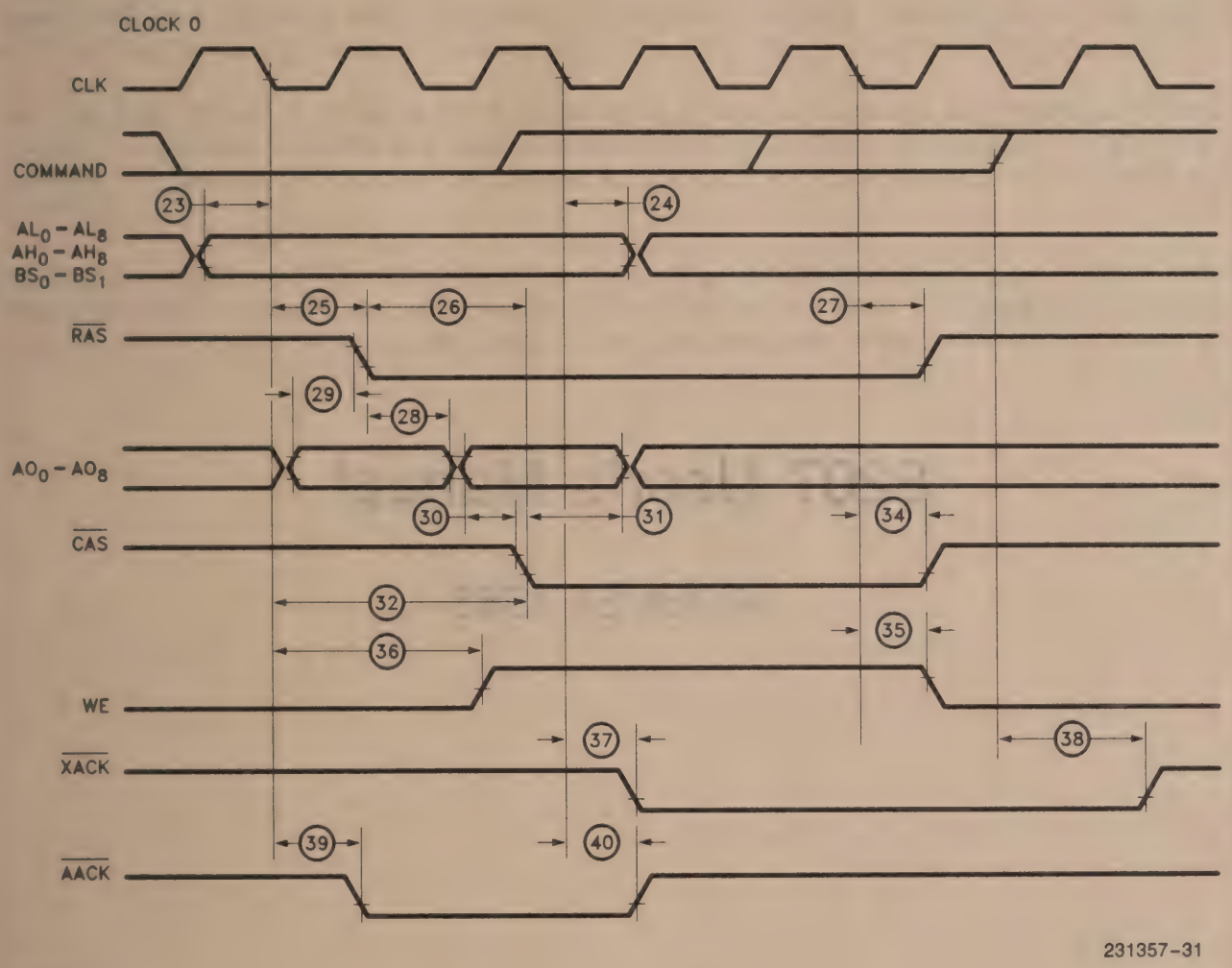
WAVEFORMS (Continued)

ASYNCHRONOUS PORT INTERFACE



WAVEFORMS (Continued)

RAM INTERFACE TIMING



NOTE:
Actual transitions are programmable. See Tables 8 and 9.

8207 User's Manual

AUGUST 1983

NOVEMBER 1983
ORDER NUMBER: 230822-002

CHAPTER 1
INTRODUCTION

This guide is a supplement to the 8207 Data Sheet[1] and is intended as a design aid and not a stand-alone description of the 8207. The reader should already have read and have a copy of the 8207 Data Sheet, 8206 Error Detection and Correction Unit Data Sheet (EDCU), a microprocessor Data Sheet, or a Multibus bus specification for interfacing to the 8207, and a dynamic RAM Data Sheet[2].

The Intel 8207 Advanced Dynamic RAM Controller is a high performance, highly integrated device designed to interface 16k, 64k, and 256k dynamic RAMS to Intel microprocessors. The 8207, with the 8206, provides complete control for memory initialization, error correction, and automatic error scrubbing.

The 8207 has several speed selected versions. The -16 and -12 parts are for clock speeds up to 16MHz and 12 MHz in "fast cycle" configurations, and up to 8 MHz and 6 MHz in "slow cycle" configurations. The -8 and -6 parts can only be used in slow cycle configurations and as a result have some relaxed A.C. timings.

NOTE:

(1) The most current Data Sheet is dated July, 1984
(2) All RAM cycle timings and references are based on Intel's 2164A Dynamic RAMs, APR '82

CHAPTER 2
PROGRAMMING THE 8207

The many configurations of bus structures, RAM speeds, and system requirements that the 8207 supports require the 8207 to be programmable. The 8207 will modify its outputs to provide the best performance possible. The 8207 must be told what type of interface the memory commands will arrive on, what type of RAM (speed, refresh rate) is being used, the clock rate, and others.

The 8207 uses two means to be informed of the user's requirements. It reads in a 16 bit serial program word and examines the logic states on several input pins. The pins that are sampled for a logic level give the user options on the types of refresh and memory command input timing.

Input Pin Options

The three input pins that configure part of the 8207 are: PCTLA, PCTLB, and REFRQ. Let's examine the options in refresh types the REFRQ pin provides.

Refresh types:

The 8207 gives the user a choice of the following refresh types.

1) Internal Refresh: All refresh cycles are generated internally — based on an internal programmable time.

2) External Refresh with Failsafe: If the external logic does not generate a refresh cycle within the programmed period, the 8207 will.

3) External Refresh - No Failsafe or No Refresh; All refresh cycles are generated at times by the user. This is for systems that cannot tolerate the random delay imposed by refresh (i.e. graphics memory).

4) Burst Refresh: The 8207 generates up to 128 consecutive refresh cycles and must be requested by external logic. Memory requests will be performed when the burst is completed.

The 8207 examines the state of the REFRQ pin when RESET goes inactive. This timing is shown in the "Clock and Programming Timings" waveforms in the Data Sheet.

If REFRQ is sampled active by the falling edge of RESET, the 8207's internal timer is enabled. The timer's period is determined by the CI0, CI1, and PLS bits in the program word. External refresh cycles are generated by a low to high transition on the REFRQ input. This transition, besides generating a refresh cycle, also resets the internal timer to zero. Simply tie REFRQ to Vcc if internal refresh is required.

If REFRQ is seen low at the falling edge of RESET, the internal timer is deactivated. All refresh cycles must either be done by external logic or by accessing all RAM (internal) rows within a 2 ms period.

Once the no failsafe option is programmed, the 8207 will generate a burst of up to 128 refresh cycles when the REFRQ input goes from low to high and sampled high for two consecutive clock edges. These cycles are internally counted and the 8207 stops when the refresh address counter reaches the value $XX1111111_2$ (X = don't care; see *Refresh Counter* section). If prior to the burst request the counter is at $XX1111110_2$ then only 2 refresh cycles would be generated.

For a single refresh cycle to be generated via external logic, the REFRQ input will have to go from low to high and then sample high by a falling 8207 clock edge. Since external refresh requests typically arrive asynchronously with respect to the 8207's clock, this requires the REFRQ to be synchronized to the 8207 clock when programmed in the failsafe mode. This is to ensure that the request is seen for one clock - no more, no less. If no external synchronization is performed, then the 8207 could do random burst cycles.

Processor Interface Options:

The PCTLA, PCTLB input pins will program the 8207 to accept either the standard demultiplexed $\overline{RD}$ and $\overline{WR}$ inputs, or to directly decode the status outputs of Intel's iAPX86, 88 family of microprocessors. The state definitions of the status lines and their timings, relative to the processor clock, differ for the 8086 family and the iAPX286 processor. Table 1 illustrates how the 8207 interprets these inputs after the PCTL pins are programmed.

If PCTL is seen high, as RESET goes inactive, and 8086 status interface is enabled. The commands arriving at the 8207 are sampled by a rising clock edge. When PCTL is low, the 80286 status and Multibus command interface is selected. These commands are sampled by the 8207 by a falling clock edge.

More information on interfacing to processors is contained in the *Microprocessor Interface section.*

Table 1. Status Coding of 8086, 80186 and 80286

Status Code			Function	
$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	8086/80186	80286
0	0	0	Interrupt	Interrupt
0	0	1	I/O Read	I/O Read
0	1	0	I/O Write	I/O Write
0	1	1	Halt	Idle
1	0	0	Instruction Fetch	Halt
1	0	1	Memory Read	Memory Read
1	1	0	Memory Write	Memory Write
1	1	1	Idle	Idle

8207 Response

8207 Command			Function	
PCTL	$\overline{RD}$	$\overline{WR}$	8086 Status Interface	Command Interface
0	0	0	Ignore	Ignore
0	0	1	Ignore	Read
0	1	0	Ignore	Write
0	1	1	Ignore	Ignore
1	0	0	Read	Ignore
1	0	1	Read	Inhibit
1	1	0	Write	Inhibit
1	1	1	Ignore	Ignore

Programming Word

The 8207 requires more information to operate in a wide variety of systems. The 8207 alters its timings and pin functions to operate with the 8206 ECC chip. The programming options allow the designer to use asynchronous or synchronous buses, various clock rates, various speeds and types of RAM, and others. This is detailed in Table 2.

This data is supplied to the 8207 over the PDI input pin. There are two methods of supplying this data. One is to strap the PDI pin high or low with the subsequent restrictions on your system. Table

3 shows the required system configuration. Note that your only option when strapping this pin high or low is error correction or not.

If any other configurations are required, then the programming data will have to be supplied by one or two 74LS165 type shift registers. Note that the sense of the bits in the program word change between ECC and non-ECC configurations.

Table 2a.
Non-ECC Mode Program Data Word

PD15								PD8 PD7							PD0
0	0	TM1	$\overline{PPR}$	$\overline{FFS}$	EXT	$\overline{PLS}$	CI0	CI1	$\overline{RB1}$	$\overline{RB0}$	$\overline{RFS}$	$\overline{CFS}$	SB	$\overline{SA}$	0

Program Data Bit	Name	Polarity/Function
PD0	ECC	ECC = 0 For non-ECC mode
PD1	$\overline{SA}$	$\overline{SA}$ = 0 Port A is synchronous $\overline{SA}$ = 1 Port A is asynchronous
PD2	SB	SB = 0 Port B is asynchronous SB = 1 Port B is synchronous
PD3	$\overline{CFS}$	$\overline{CFS}$ = 0 Fast-cycle iAPX 286 mode $\overline{CFS}$ = 1 Slow-cycle iAPX 86 mode
PD4	$\overline{RFS}$	$\overline{RFS}$ = 0 Fast RAM $\overline{RFS}$ = 1 Slow RAM
PD5 PD6	$\overline{RB0}$ $\overline{RB1}$	RAM bank occupancy See Table 4
PD7 PD8	CI1 CI0	Count interval bit 1: see Table 6 in 8207 data sheet Count interval bit 0: see Table 6 in 8207 data sheet
PD9	$\overline{PLS}$	$\overline{PLS}$ = 0 Long refresh period $\overline{PLS}$ = 1 Short refresh period
PD10	EXT	EXT = 0 Not extended EXT = 1 Extended
PD11	$\overline{FFS}$	$\overline{FFS}$ = 0 Fast CPU frequency $\overline{FFS}$ = 1 Slow CPU frequency
PD12	$\overline{PPR}$	$\overline{PPR}$ = 0 Most recently used port priority $\overline{PPR}$ = 1 Port A preferred priority
PD13	TM1	TM1 = 0 Test mode 1 off TM1 = 1 Test mode 1 enabled
PD14	0	Reserved must be zero
PD15	0	Reserved must be zero

Table 2b
ECC Mode Program Data Word

PD15							PD8 PD7							PD0
$\overline{TM2}$	RB1	RB0	PPR	FFS	$\overline{EXT}$	PLS	$\overline{CI0}$	$\overline{CI1}$	XB	$\overline{XA}$	RFS	CFS	$\overline{SB}$	SA	1

Program Data Bit	Name	Polarity/Function	
PD0	ECC	ECC = 1	ECC mode
PD1	SA	SA = 0 SA = 1	Port A is asynchronous (late AACK) Port A is synchronous (early AACK)
PD2	$\overline{SB}$	$\overline{SB}$ = 0 $\overline{SB}$ = 1	Port B is synchronous (early AACK) Port B is asynchronous (late AACK)
PD3	CFS	CFS = 0 CFS = 1	Slow-cycle iAPX 86 mode Fast-cycle iAPX 286 mode
PD4	RFS	RFS = 0 RFS = 1	Slow RAM Fast RAM
PD5	$\overline{XA}$	$\overline{XA}$ = 0 $\overline{XA}$ = 1	Multibus-compatible XACKA AACKA not multibus-compatible
PD6	XB	XB = 0 XB = 1	AACKB not multibus-compatible Multibus-compatible XACKB
PD7 PD8	CI1 CI0	Count interval bit 1: see Table 6 in 8207 data sheet Count interval bit 0: see Table 6 in 8207 data sheet	
PD9	PLS	PLS = 0 PLS = 1	Short refresh period Long refresh period
PD10	$\overline{EXT}$ $\overline{EXT}$	$\overline{EXT}$ = 0 $\overline{EXT}$ = 1	Master and slave EDCU Master EDCU only
PD11	FFS	FFS = 0 FFS = 1	Slow CPU frequency Fast CPU frequency
PD12	PPR	PPR = 0 PPR = 1	Port A preferred priority Most recently used port priority
PD13 PD14	RB0 RB1	RAM bank occupancy See Table 4	
PD15	TM2	TM2 = 0 TM2 = 1	Test mode 2 enabled Test mode 2 off

Table 3. 8207 Default Programming

Port A is Synchronous—has early AACK
Port B is Asychronous—has late AACK
Fast RAM
Refresh Interval uses 236 clocks
128 Row refresh in 2 ms; 256 Row refresh in 4 ms
Fast Processor Clock Frequency (16 MHz)
"Most Recently Used" Priority Scheme
4 RAM banks occupied

Reset

If Port A is changed to an asynchronous interface (via the SA bit), then one of two precautions must be taken. Either a differentiated reset must be provided, or else software must not access the 8207 controller RAM for a short period. The 8207 is either adding or deleting internal synchronizing circuits. If a command is received during this changing, the 8207 may not perform properly. This is required only if Port A is changed to asynchronous, or if Port B is changed to synchronous.

Several of the bits in the program word determine a particular configuration of the 8207 (reference Tables 10, 11 and the 8207 Data Sheet). The bits are: CFS, CLOCK fast or slow; RFS, RAM access time fast or slow (fast refers to 100 ns - slow is everything greater); and EXT, for memory data word widths greater than 16 (22) bits. Generally speaking, CO is the fastest configuration at clock frequencies up to 16 MHz, both in the ECC or non-ECC charts. 'C3' is the fastest for 8 MHz clocks in non-ECC mode, and 'C4' is the fastest configuration when using ECC.

Take, for example, a 16 MHz 8207 clock with no error correction, a 16 bit word, and 150 ns (slow) dynamic RAMs. Table 10, in the 8207 data sheet, is used to arrive at the configuration "C1." The Timing chart Table 12 in the 8207 Data Sheet is then used to determine which clock edge to reference all timings from. The Waveforms diagrams then are used to determine the delay from the clock edge.

CHAPTER 3
RAM INTERFACE

The 8207 takes the memory addresses from the microprocessor bus and multiplexes them into row and column addresses as required by dynamic RAMs. The only hardware requirement when interfacing the 8207 to dynamic RAM are series resistors on all the RAM outputs of the 8207, and proper layout of the traces (see Intel's RAM Data Sheets or the Memory Design Handbook). This section mainly details the effects and requirements of input signals to the 8207 on the RAM array.

The 8207 contains an internal address counter used for refresh and error scrubbing (when using the 8206 EDCU) cycles. The 8207 has 18 address inputs (AIL0-AIL8 and AIH0-AIH8) which are multiplexed to form 9 address outputs (A00-A08). There are also 2 bank select (BS0, BS1) inputs for up to 4 banks of RAM. The Bank Select inputs are decoded internally to generate $\overline{RAS}$ and $\overline{CAS}$ outputs.

Refresh Interval

The 8207 supports four different refresh techniques as described in the *Refresh Options* section. In addition, the rate at which refresh cycles are performed is programmable. This is necessary because the refresh period is generated from the CLK input, which may vary over a wide range of frequencies. Programming the Cycle Fast/Slow (CFS) and Frequency Fast/Slow (FFS) bits automatically reprograms the refresh timer to generate the correct refresh interval for a clock frequency of 16, 10, 8, or 5 MHz (CFS, FFS = 11, 10, 01, and 00, respectively). For clock frequencies between those, Count Interval (CI1, CI0) programming bits allow "fine tuning" of the refresh interval. Refresh will always be done often enough to satisfy the RAM's requirements without doing refresh more often than needed and wasting memory bandwidth for all clock frequencies.

Refresh Counter

The internal refresh address counter of the 8207 contains 20 bits as organized in Figure 1.

19 18	17 16 15 14 13 12 11 10 9	8 7 6 5 4 3 2 1 0
Bank	Col addr	Row addr

Figure 1. 8207 Refresh Address Counter

In non-ECC mode, the refresh address counter does not count beyond bit 8. For standard RAMs, this will refresh 128 rows every 2 ms or 256 rows every 4 ms.

In ECC mode, the 8207 automatically checks the RAM for errors during refresh. This requires it to access each of the possible 2^{20} words of memory. The 8207 does not delete any of these bits when used with 16k and 64k dynamic RAMs. Each column would be scrubbed 4 times with 16k RAMs, and twice with 64 RAMs. This will have no detrimental effect on reliability. Banks of RAM that are not occupied, as indicated to the 8207 by the RB0, RB1 programming bits, will not be scrubbed.

Bank Selects BS0, BS1; RB0, RB1

The 8207 is designed to drive up to 88 RAMs in various configurations. The 8207 takes 2 inputs, BS0, BS1, and decodes them based on 2 programming bits, RB0, RB1, to generate the required $\overline{RAS}$/$\overline{CAS}$ strobes. Additionally, the 8207 will always recognize (not programmable) whether an access is made to the same RAM bank or to a different bank. The 8207 will interleave the accesses resulting in improved performance.

RAS and CAS Reallocation

The 8207's address lines are designed to drive up to 88 RAMs directly (through impedance matching resistors). The 4 $\overline{RAS}$ and $\overline{CAS}$ outputs drive up to 22 RAMs per bank (16 data plus 6 check bits with the 8206). Under these conditions, the 8207 will meet all RAM timing requirements. See Figure 2 for an example.

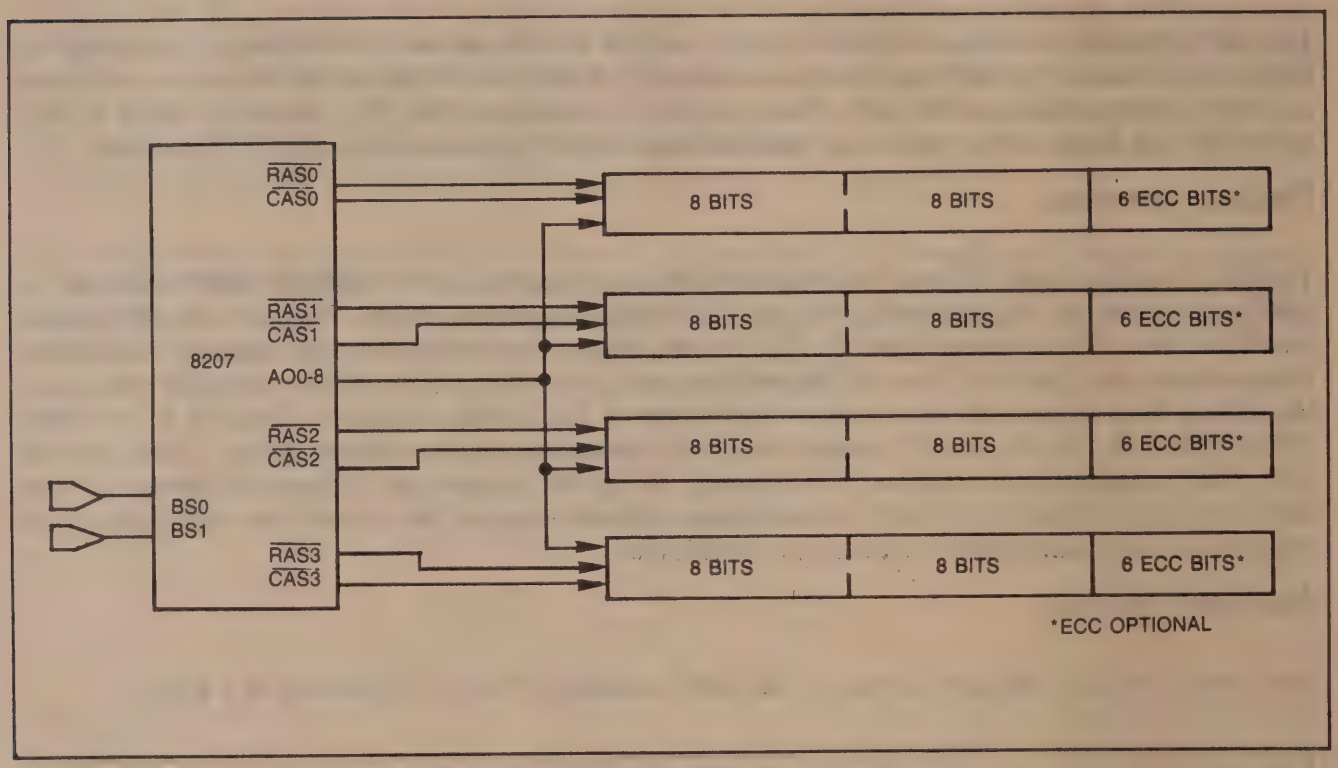


Figure 2. 8207 4 RAM Bank Configuration

The 8207 can accommodate other configurations like a 32 bit error corrected memory system. Each bank would have 39 RAMs (32 + 7 check bits) with the total number of RAMs equal to 78. This is within the address drivers capability, but the 39 RAMs per bank exceeds the $\overline{RAS}$ and $\overline{CAS}$ drivers limits. The loading of the $\overline{RAS}/\overline{CAS}$ drivers should not exceed 22 RAMs per bank, otherwise critical row, column address setup, and hold times would be violated.

In order to prevent these critical timings being violated, the 8207 will re-allocate the $\overline{RAS}$ and $\overline{CAS}$ drivers based on the RB0, RB1 programming bits (see Table 4). If the RB0, RB1 bits are programmed for 2 banks, the 8207 will operate $\overline{RAS0}$ and $\overline{RAS1}$ as a pair along with $\overline{RAS2}$ and $\overline{RAS3}$, $\overline{CAS0}$ and $\overline{CAS1}$, and $\overline{CAS2}$ and $\overline{CAS3}$. Now the address drivers would be loaded by 78 RAMs and the $\overline{RAS}/\overline{CAS}$ drivers by 20 RAMs. This relative loading is almost identical to the first case of four banks of 22 RAMs each. Drive reallocation allows a wide range of memory configurations to be used and still maintain optimal memory timings. Figure 3 shows a 32 bit non-error corrected configuration.

These programming bits do not help to qualify RAM cycles. Their purpose is to reallocate $\overline{RAS}/\overline{CAS}$ drivers. For example, if there is one bank of RAM and the bank select inputs (BS0, BS1) select any other bank and no provision is made to deselect the 8207 (via PE), the 8207 will do a RAM cycle and issue an acknowledge. This happens irregardless of the RB0, RB1 programmed value. See the *Optional RAM Bank's* section to provide for this.

Table 4. RAM Bank Selection Decoding and Word Expansion

Program Bits RB1	RB0	Bank Input B1	B0	$\overline{RAS}/\overline{CAS}$ Pair Allocation
0	0	0	0	$\overline{RAS}_{0\text{-}3}$, $\overline{CAS}_{0\text{-}3}$ to Bank 0
0	0	0	1	Illegal Bank Input
0	0	1	0	Illegal Bank Input
0	0	1	1	Illegal Bank Input
0	1	0	0	$\overline{RAS}_{0,1}$, $\overline{CAS}_{0,1}$ to Bank 0
0	1	0	1	$\overline{RAS}_{2,3}$, $\overline{CAS}_{2,3}$ to Bank 1
0	1	1	0	Illegal Bank Input
0	1	1	1	Illegal Bank Input
1	0	0	0	$\overline{RAS}_0$, $\overline{CAS}_0$ to Bank 0
1	0	0	1	$\overline{RAS}_1$, $\overline{CAS}_1$ to Bank 1
1	0	1	0	$\overline{RAS}_2$, $\overline{CAS}_2$ to Bank 2
1	0	1	1	Illegal Bank Input
1	1	0	0	$\overline{RAS}_0$, $\overline{CAS}_0$ to Bank 0
1	1	0	1	$\overline{RAS}_1$, $\overline{CAS}_1$ to Bank 1
1	1	1	0	$\overline{RAS}_2$, $\overline{CAS}_2$ to Bank 2
1	1	1	1	$\overline{RAS}_3$, $\overline{CAS}_3$ to Bank 3

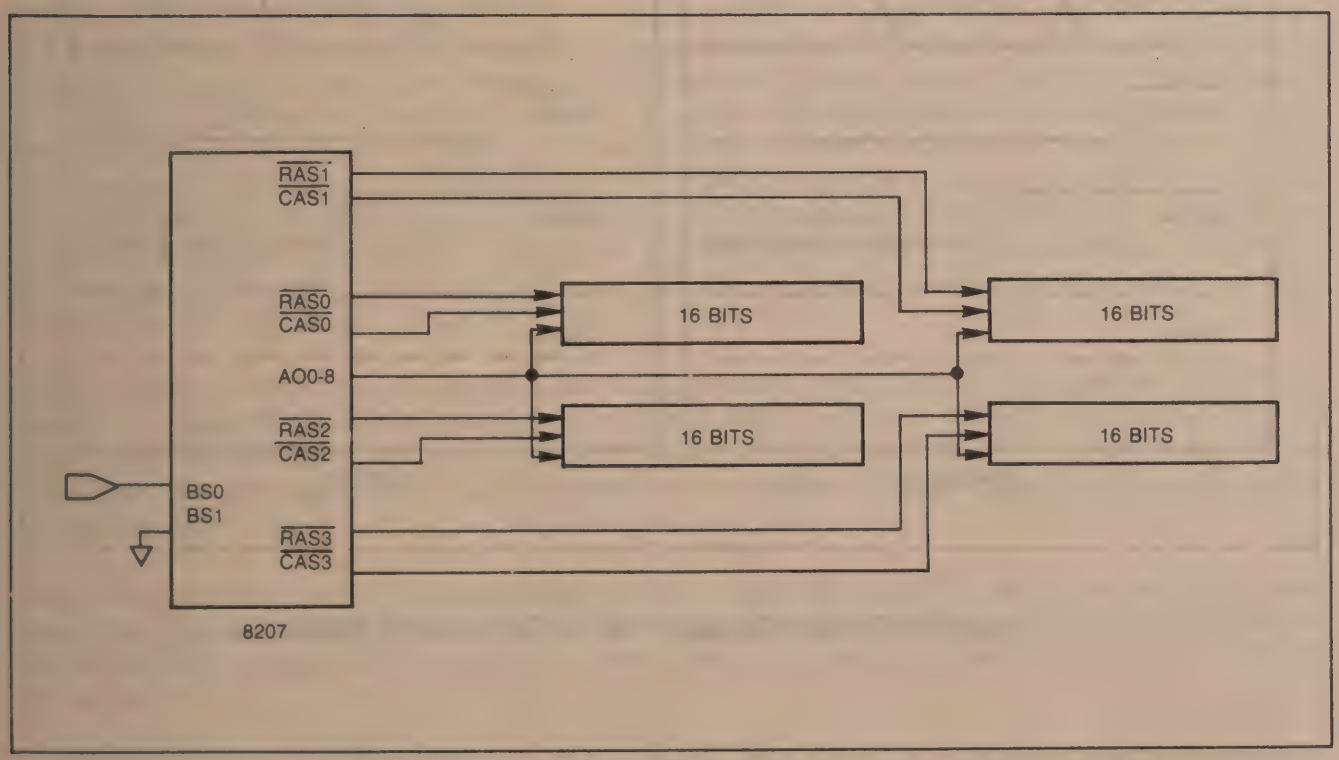


Figure 3. 8207 2 RAM Bank Configuration

Scrubbing

An additional function of the RB0, RB1 bits, besides $\overline{RAS}/\overline{CAS}$ allocation, is to inform the 8207 of how many banks are physically present. The 8207 will, during the refresh cycle, read data from a location and check to see that data and check bits are correct. If there is an error, the 8207 lengthens the refresh cycle and writes the corrected data back into RAM. Scrubbing the entire memory greatly reduces the chance of an uncorrectable error occurring. See the ***Refresh*** section for more detail on scrubbbing.

Refresh Cycles

The 8207 performs $\overline{RAS}$ only refresh cycles in non-ECC systems. It outputs all 8207 control signals except for $\overline{CAS}$ and acknowledges. The real delay in a system due to refresh would be a fraction of that value[1]. The length of the refresh cycle is always 2tRP + tRAS, and varies based upon the programmed 8207 configuration.

In error-corrected systems, the refresh cycle is actually a read cycle. The 8207 outputs a row address, then all $\overline{RAS}$ outputs go active. Next, a column address is output and then $\overline{CAS}$. The $\overline{CAS}$ output is based upon the RB0, RB1 allocation bits. Figure 4a shows the general timing for a four bank system, and Figure 4b shows a two bank system.

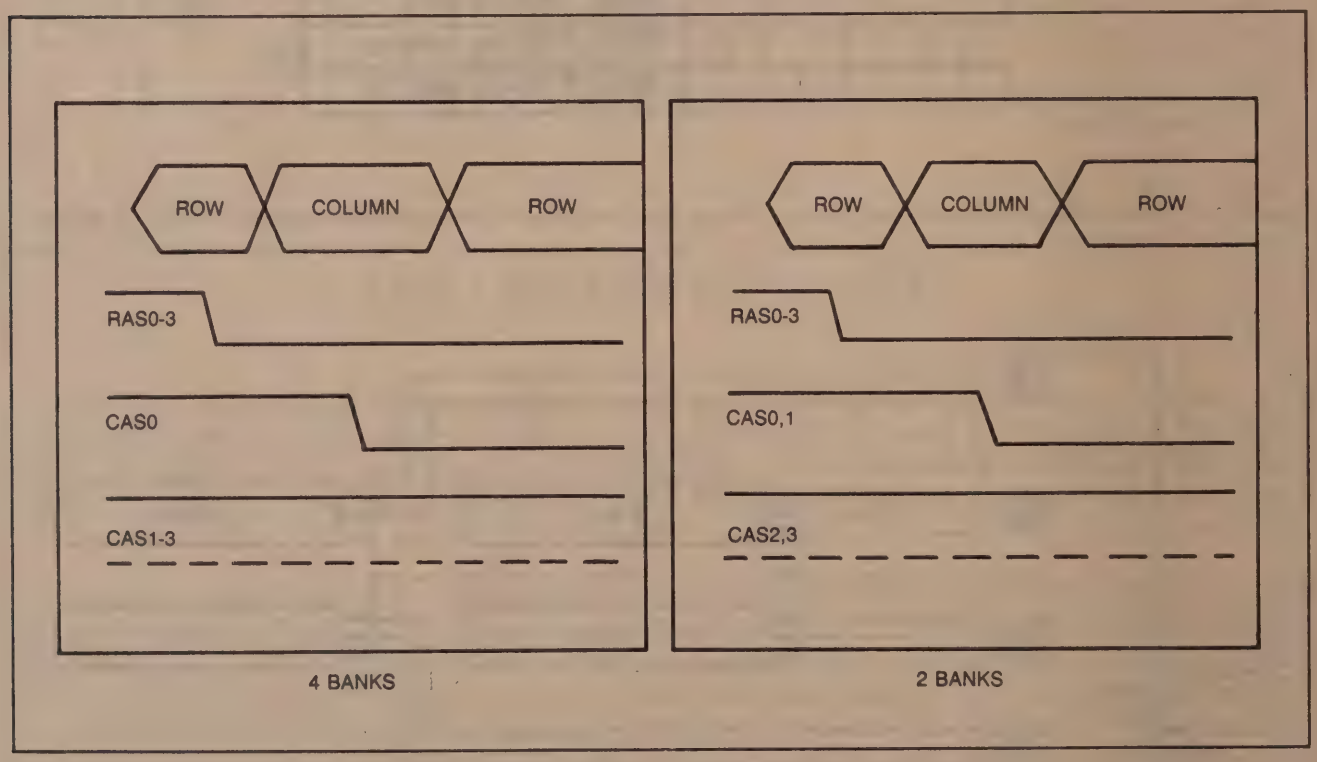


Figure 4. Refresh Cycles for Error Corrected Systems

(1) Measurements have shown a delay of 2-4% on program execution time compared to programs running without refresh.

The 8207 sends the read out word through the 8206 EDCU to check for any errors. If no errors, the refresh cycle ends. If an error is discovered, the 8207 lengthens the cycle. An error is determined if the ERROR output of the 8206 is seen active at the same edge that the 8207 issues the R/W output. The cycle is then lengthened to a RMW cycle. If the error was correctable, the corrected data is written back to the location it was read from. But, if the data is uncorrectable, the cycle is still lengthened to a RMW, but no write pulse is issued. To aid in stabilizing the RAM output data and the Error flag, pullup resistors of 10k ohms on the data out lines are recommended.

Scrubbing removes soft errors that may accumulate until a double-bit error occurs, which would halt the system. Hard single-bit failures will not stop the system, but could slow it down. This is because read and refresh cycles lengthen to correct the data.

For large RAM arrays some form of error logging or diagnostics should be considered.

Interleaving

The term "interleaving" is often used to refer to overlapping the cycle times of multiple banks (or boards or systems) of RAMs. This has the advantage of using relatively slow cycle time banks to achieve a faster perceived cycle time at the processing unit. The drawbacks of interleaving are more logic to handle the necessary control and, for maximum performance, the program should execute sequentially through the addresses.

Dynamic RAM cycles consist of 2 parts — the RAS active time (tRAS in Dynamic RAM Data Sheets) and precharge time (tRP). The sum of these two times are roughly equal to the cycle time of the RAM. The 8207 determines how long these two periods are, based on the configuration the user picked (via the programming bits). Bank interleaving, as used by the 8207, is slightly different than the previous definition. The 8207 will overlap the precharge time of one bank with the access time of another bank. In either case, the advantage is the effective cycle time is reduced without having to use faster RAMs.

For interleaving to take place there must be more than 1 bank of RAM connected to the 8207. Interleaving is not practical with 3 banks of RAM because 3 is not a power of 2 (the 2 bank inputs BS0, BS1). So, interleaving works only for 2 or 4 banks of RAM. Note that it is easy enough to use three banks of RAM where the bank select inputs are connected to the highest-order address line. For instance, if three banks of 2164s are used in an 8086 system, and located at address OH, bank selects BS0 and BS1 would be connected to microprocessor addresses A17 and A18, respectively. Banks 0-2 would be accessed in the address ranges OH - FFFFH, 10000H - 1FFFFH, and 20000H - 2FFFFH, respectively. In this case, consecutive addresses are almost always in the same bank and very little interleaving can take place.

Figure 5 shows the effects on the performance of the processor with and without interleaving. In both examples, consecutive accesses to the same bank will add 1 wait state to the second access, but no wait states to consecutive accesses to different banks. Irregardless of the 8207 configuration, there will always be a minimum 1 wait state added without interleaving. Therefore, interleaving is very highly recommended!

Interleaving is accomplished by connecting the 8207's BS0, BS1 inputs to the microprocessor's low order word address lines. Each consecutive address is then located in a different bank of RAM. About 90% of memory accesses are sequential, so interleaving will occur about 90% of the time in a single port system.

In a dual port system, the advantages of interleaving are a function of the number of banks of memory. Since the memory accesses of the two ports are presumably independent, and both ports are continuously accessing memory, the 8207 arbiter will tend to interleave accesses from each port (i.e., Port A, Port

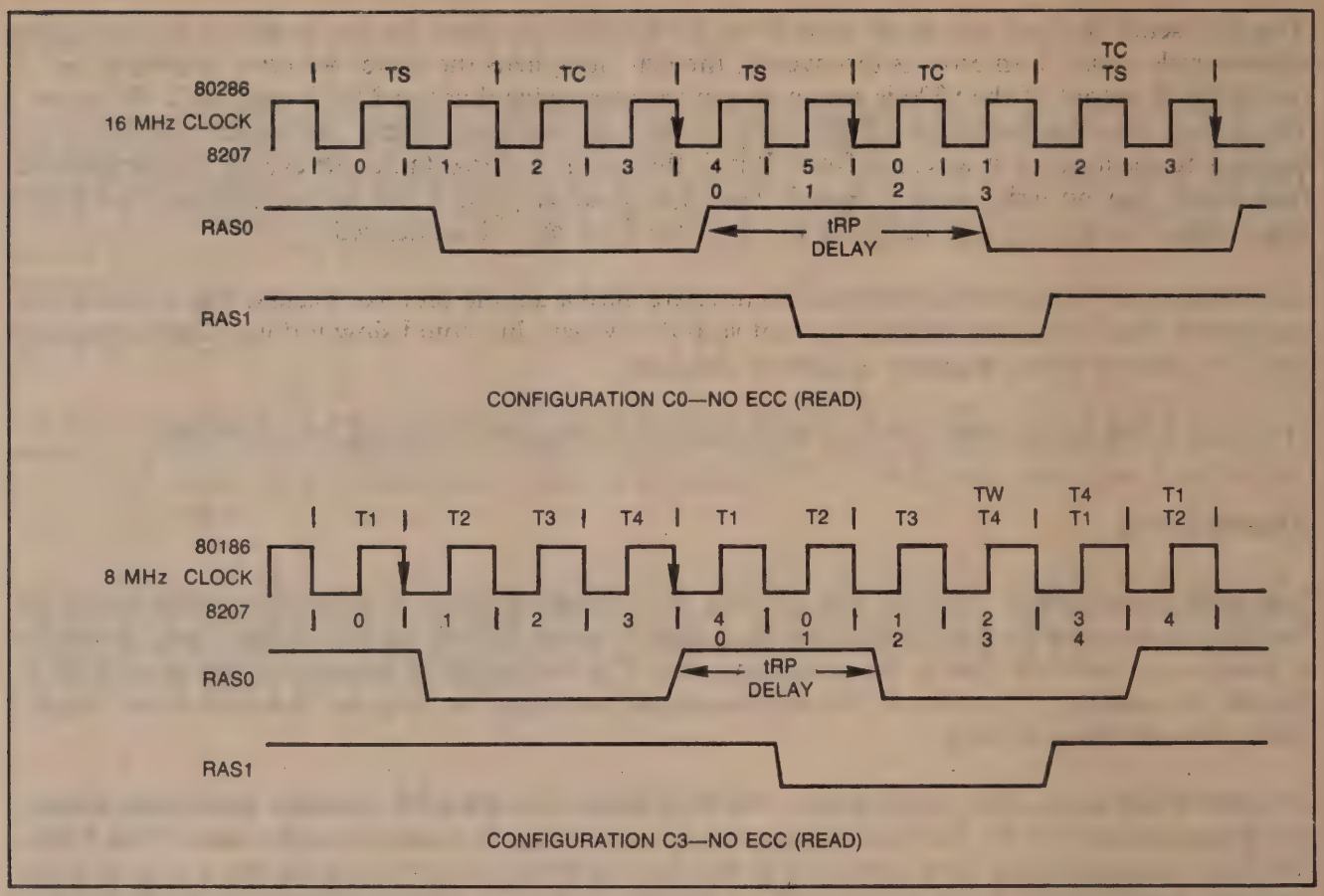


Figure 5. Processor Performance With and Without Interleaving

B, Port A, Port B, ...). If there are two banks of RAM interleaving will occur 50% of the time and, if there are four banks of RAM, interleaving will take place 75% of the time[1]. To the extent that a single port generates a majority of memory cycles, interleaving efficiency will approach 90% as described in the previous paragraph.

(1) Don't get confused here. The paragraph is talking about interleaving memory requests from both ports, and their probability of accessing one of the other banks of RAM where tRP has been satisfied. The 8207 will leave the RAM precharge time out if consecutive accesses go to different banks. The 8207 RAM timing logic does not care which port requests a RAM cycle. requests a RAM cycle.

Optional RAM Banks

Many users allow various RAM array sizes for customer options and future growth. Some care must be taken during the design to allow for this. Three items should be considered to permit optional RAM banks.

The first item is the total RAM size. The 8207 starts a memory cycle based only upon a valid status or command and $\overline{PE}$ active. So some logic will be required to deselect the 8207 (via $\overline{PE}$) when the addressed location does not exist within the current memory size. A 7485 type magnitude comparator works well.

The second item to consider is the BS0, BS1 inputs. With one bank of RAM these inputs are tied to ground. Four banks of RAM require two address inputs. So, if the design ever needs four banks

of RAM, then the BS0, BS1 inputs must be connected to address lines. Selecting a non-existant RAM bank is illegal. Figure 6 shows a non-interleaved method.

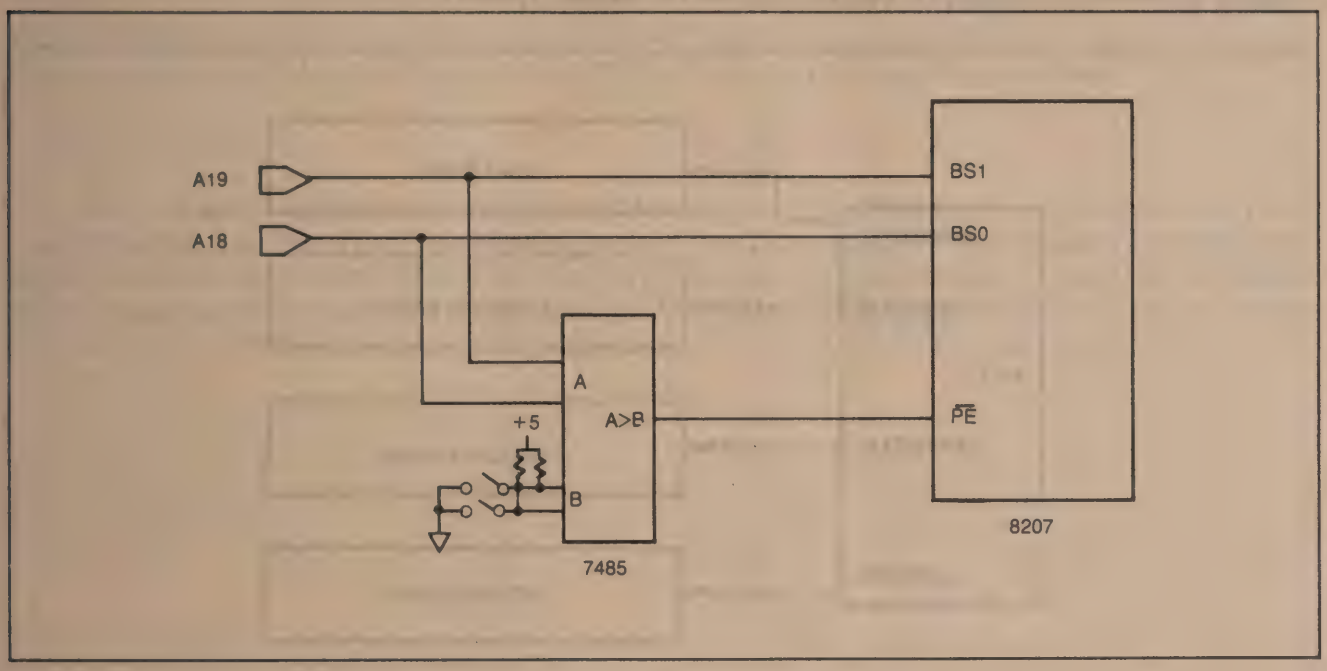


Figure 6. Non-Interleaved 8207 Selection Circuit

With designs using interleaving, the least significant word address lines are connected to the BS0, BS1 inputs. With two banks of RAM, A1 from the Intel processor is connected to BS0. A2 is connected to BS1, but not allowed to function until four banks are present. However, A2 must still be used since addresses increase sequentially. Two possible ways of implementing this are shown in Figure 7 below.

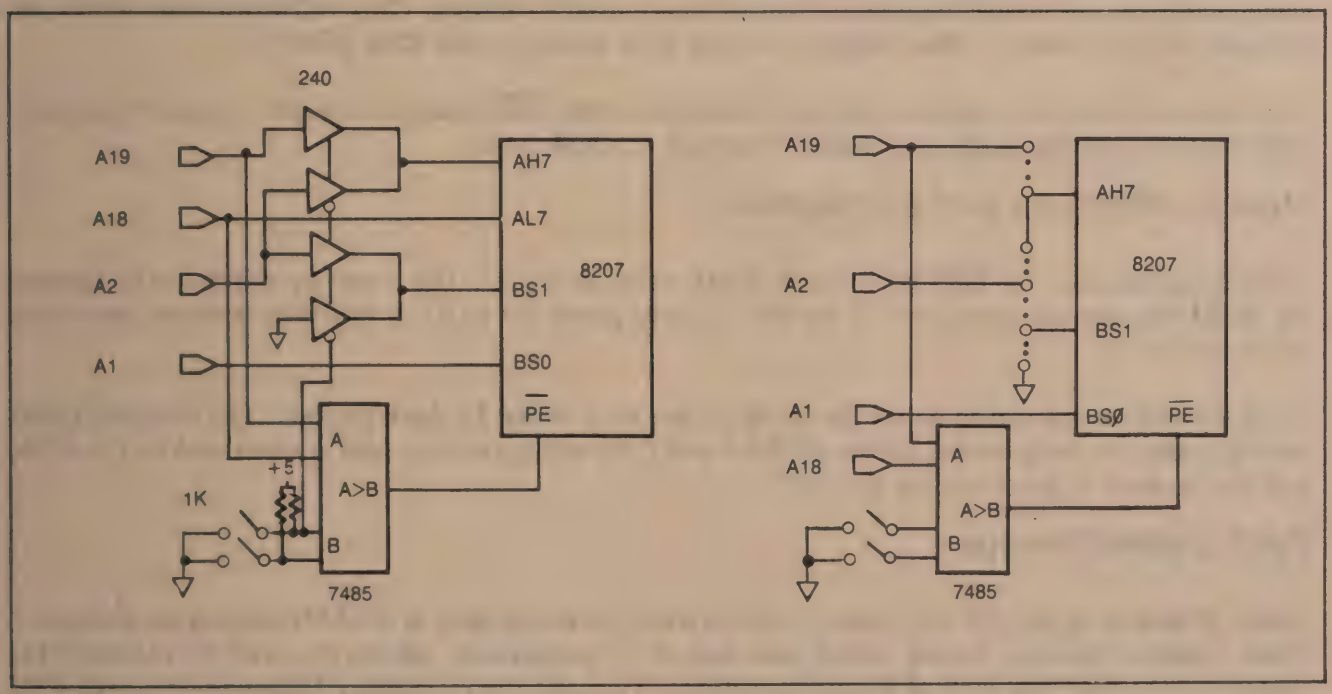


Figure 7. Interleaved 8207 Selection Circuits

The final consideration is for the $\overline{RAS}/\overline{CAS}$ outputs. Remember that when the RB0, RB1 bits are programmed for two banks, then $\overline{RAS0}$, 1 operates in tandem (non-ECC mode/ECC mode - the $\overline{CAS}$ outputs also work in tandem). Figure 8 shows the proper layout.

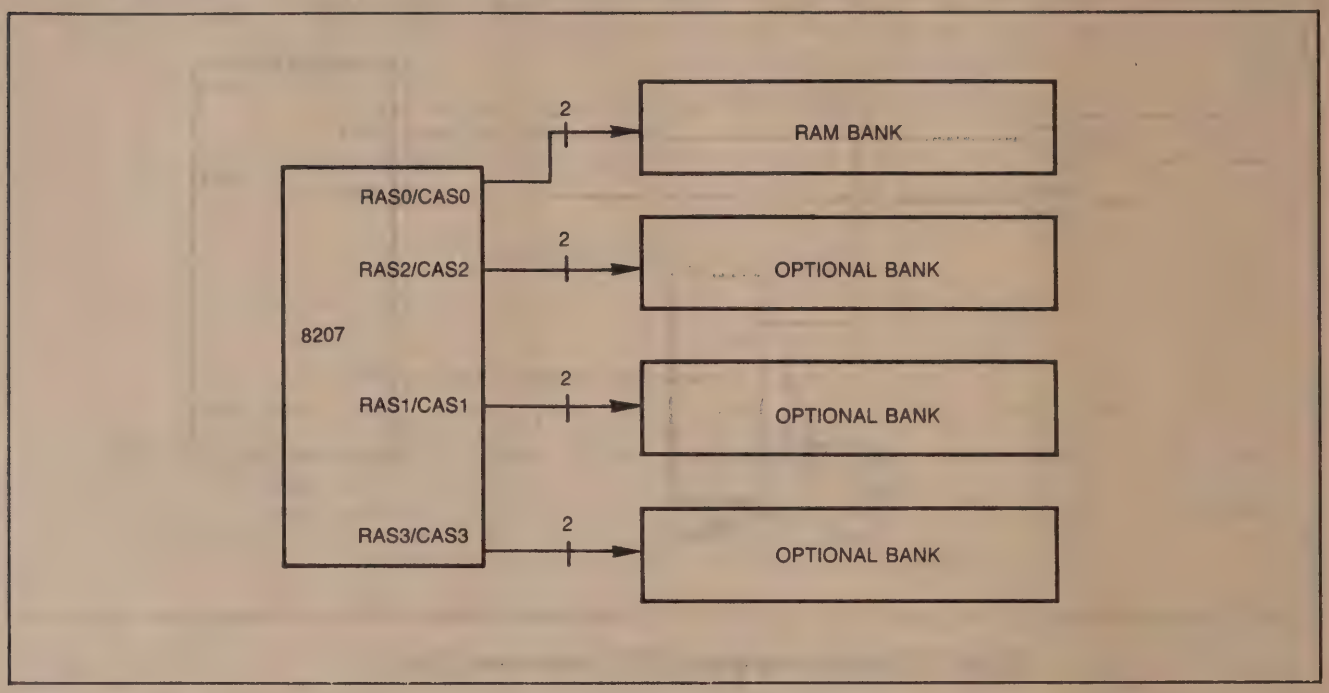


Figure 8. RAM Bank Layout

Write Enables - Byte Marks

The write enable supplied by the 8207 cannot drive the RAM array directly. It is intended to be NAND with the processor supplied byte marks in a non-ECC system. In error-corrected systems, the write enable output should be inverted before being used by RAMs. Only full word read/writes are allowed in ECC systems. The changing of byte data occurs in the 8206 EDCU.

For single and dual port systems, the byte mark data (A0, $\overline{BHE}$) must be latched. The 8207 can (and will) change the input addresses midway through a RAM cycle.

Memory Warm-up and Initialization

After programming, the 8207 performs 8 RAM warm-up cycles. The warm-up cycles are to prepare the RAMs for proper operation. If the 8207 is configured for ECC, it will then prewrite zeros into the entire array.

All RAS outputs are driven active for these cycles, once every 32 clock periods. The prewrite cycles are equivalent to write cycles, except all RAS and CAS will go active, data is generated by the 8206, and the address is generated by the 8207.

RAM Cycles/Timings

Tables 12 and 13 of the 8207 Data Sheet show on what clock edge each of the 8207 outputs are generated. This, together with the timing waveforms and A.C. parameters, allows the user to calculate the timings of the 8207 for each of its configurations. To make the job easier, Tables 14-18 of the 8207 Data Sheet precalculate dynamic RAM timings for each 8207 configuration and type of cycle. All that is required is to plug in numerical values for the 8207 parameters.

Write Cycles

The 8207 always issues WE after $\overline{CAS}$ has gone valid. These types of cycles are known as "late writes." The 8207 does this primarily to interface to the iAPX286 processor bus timings. Late writes require separate data in and data out traces to the RAM array, plus the additional drivers.

Data Latches

The 8207 is designed to meet data setup and hold times for the iAPX86 family processors when using a synchronous status interface (see Microprocessor Interface section). Other types of interfaces will require external data latches. This is because the $\overline{CAS}$ pulse is a fixed length - the user has no control (besides programming options) over lengthening $\overline{CAS}$. When $\overline{CAS}$ goes inactive, data out of the RAMs will disappear. Asynchronous interfaces should use $\overline{XACK}$ or $\overline{LAACK}$ to latch the data.

CHAPTER 4
MICROPROCESSOR INTERFACES

The 8207 is designed to be directly compatible with all Intel iAPX86, 186, 188, and 286 processors. For maximum performance, the 8207 will directly decode the status lines and operate off of the processor's clock. Additionally, the 8207 interfaces easily to other bus types that support demultiplexed address and data with separate read and write strobes.

Bus Interfaces

The 8207 easily supports either an asynchronous or synchronous command timing. The command timing can also be adjusted for various processors via the PCTL pin.

MEMORY COMMANDS

There are four inputs for each port of the 8207 that initiate a memory cycle. The input pins are $\overline{WR}$, $\overline{RD}$, PCTL, and $\overline{PE}$. The first three inputs connect directly to the iAPX 86, 88, 186, 188 $\overline{S0}$-$\overline{S2}$ outputs, respectively. For the 80286, the same connections are used except that PCTL is tied to ground. In all configurations $\overline{PE}$ is decoded from the address bus. Multibus type commands use the same input setup as the 80286.

COMMAND/STATUS INTERFACE

The status interface for the 80186 and the 80286 differ both in timing and meaning. The 8207 can be optimized for either processor by programming the PCTL input pin at RESET time. $\overline{S2}$ in 80186 systems, connects directly to PCTL. When the processor is reset it drives $\overline{S2}$ high for one clock, then tristates it. A pullup resistor to +5 will program the PCTL input for the 80186 status interface when RESET goes inactive. A pullup is required only if no component has this pullup internally.

To optimize the 8207 for the 80286 interface, PCTL is tied to ground and not used in 80286 systems. Multibus commands are similar in meaning to the 80286 status interface, and are programmed the same way. In Multibus type systems, PCTL can be used as an inhibit to allow shadow memory. PCTL would be driven high, when required, to prevent the 8207 from performing a memory cycle. It would be connected to the Multibus INH pin through an inverter.

SYNCHRONOUS/ASYNCHRONOUS COMMANDS

Each port of the 8207 can be configured to accept either a synchronous or asynchronous (via programming bits) memory request. Minimum memory request decode time (and maximum performance) is achieved using a synchronous status interface. This type of interface to the processor requires no logic for the user to implement.

An asynchronous interface is used with Multibus bus interfaces when the setup and hold times of the memory commands cannot be guaranteed. Synchronizers are added to the inputs and will require up to two clocks for the 8207 to recognize the command. It should be obvious that better performance will result if the 8207's clock is run as fast as possible.

Figure 2 of the 8207 Data Sheet shows various combinations of interfaces. The additional logic for the asynchronous interfaces is used to either lengthen the command width, to meet the minimum 8207 spec, or to make sure the command does not arrive too soon before the address has stabilized.

PORT ENABLE

The $\overline{PE}$ inputs serve to qualify a memory request. A RAM cycle, once started, cannot be stopped. A RAM cycle starts if $\overline{PE}$ is seen active at the proper clock edge and a valid command is recognized. If $\overline{PE}$ is activated after a command has gone active and inactive, no cycle will start.

Types of logic that work well are 74138 and 7485. $\overline{PE}$ should be valid as much as possible before the command arrives because, as the address bus switches and settles, glitches on $\overline{PE}$ could either: disqualify a memory cycle; delay a memory cycle; or start a memory cycle when none should have. Refer to the Port Interface Waveforms in the Data Sheet. If Port Enable is not seen active by the next or same clock edge, no memory cycle will occur unless the command is removed and brought active again.

Back to Back Commands

Holding the $\overline{RD}$, $\overline{WR}$ inputs active will not generate continuous memory cycles. Memory commands must go inactive for at least one clock period before another memory request at that port will be considered valid. Holding the inputs active will not keep the other port from gaining access to the RAM. The only signal that can prevent the other port's gaining access to the RAM is LOCK.

Address Inputs (And LOCK)

Two pins control the address inputs on the 8207, MUX and LEN. Neither are used for single port 8086 based systems. MUX is used for dual port configurations, and LEN is used for single and dual port 80286 based systems. MUX is used to gate the proper ports addresses to the 8207. If the output is high, Port A is selected. If it is low, Port B is selected.

The cross coupled NAND gates, shown in the 8207 Data Sheet (Figure 3), are used to minimize contention when switching address buses. Use of a single inverter would have both outputs enabled simultaneously for a short period. The cross coupled hand gates allow only one output enabled.

MUX also allows the single LOCK input to be multiplexed between ports. Figure 9 shows how to multiplex the LOCK input for dual port systems. See the LOCK section for more information.

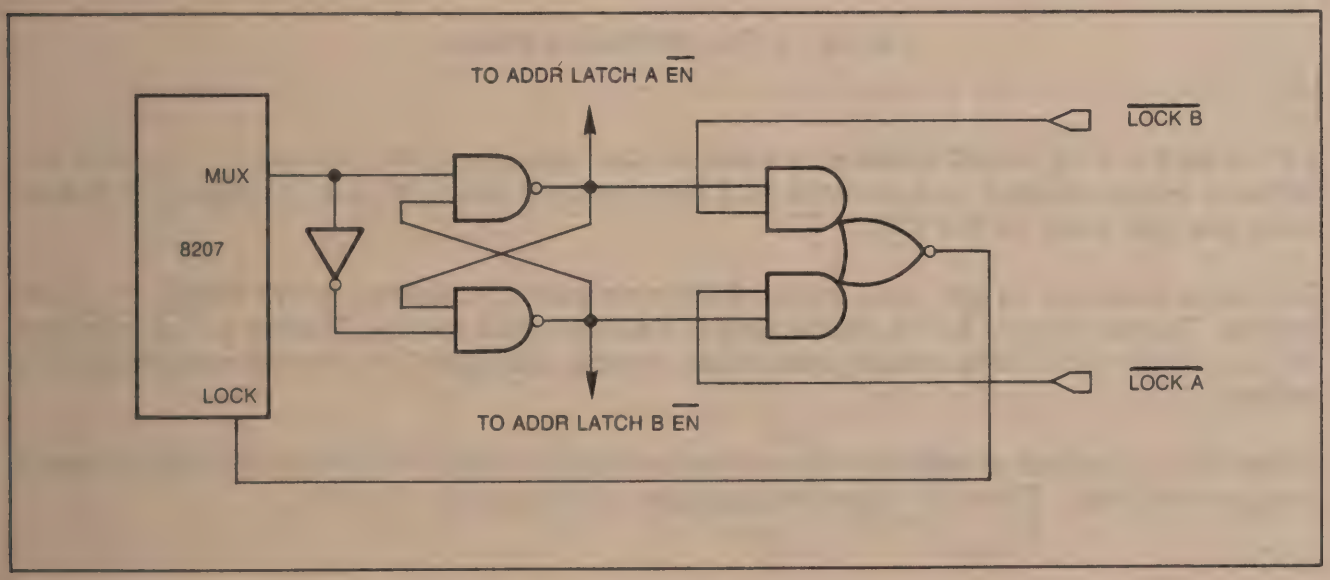


Figure 9. Dual Port LOCK Input Circuit

MUX TIMING

The MUX output is optimized by the Port Arbitration scheme, which is selected in the program word. Figure 10 shows the effects on memory selected in the program word. Figure 10 shows the effects on memory bandwidth with the different schemes. Port A Preferred optimizes consecutive cycles for Port A. Consecutive Port B cycles have at least 1 clock added to their cycle time. There would be no MUX delays for any Port A request.

The Most Recently Used scheme allows either port to generate consecutive cycles without any MUX delays. The first memory cycle for each port would have the 1 clock delay. But all others would not.

With either scheme, if both ports request the memory at their top speed, the 8207 will interleave the requests; Port A, Port B, Port A, Refresh, Port B.

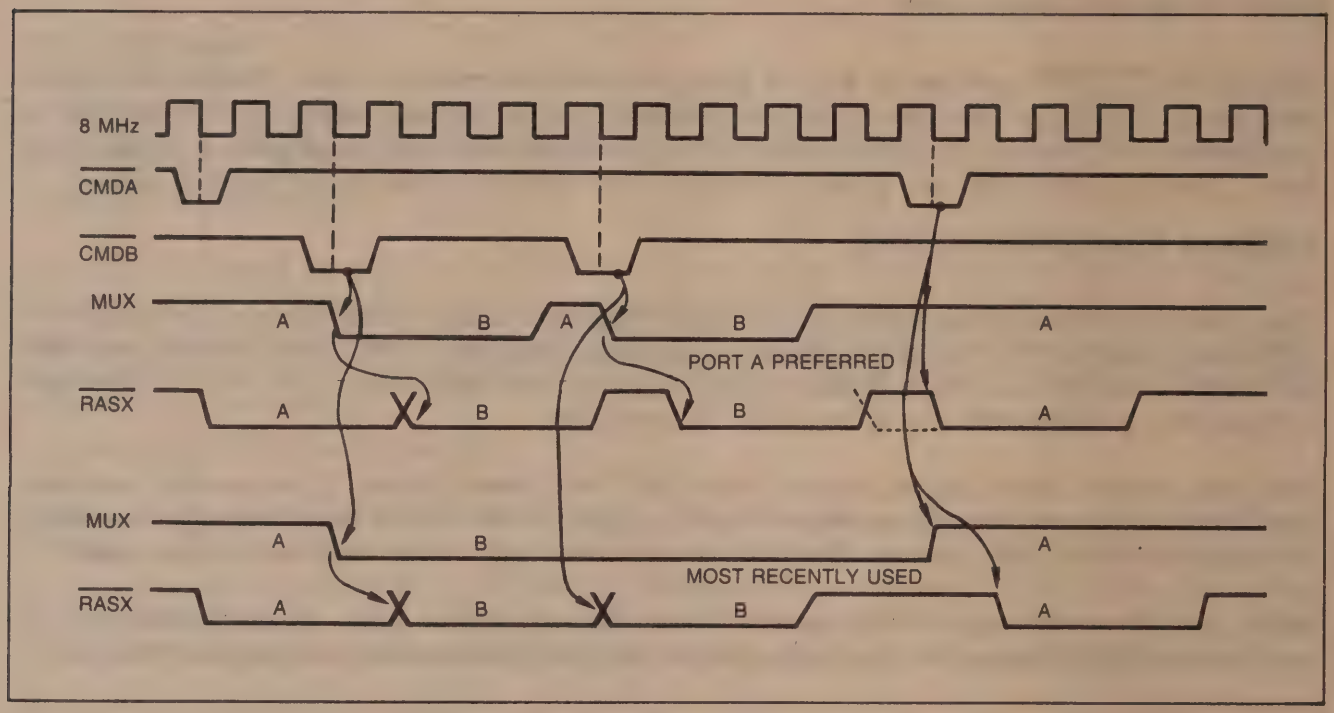


Figure 10. Port Arbitration Effects

LEN

LEN is used to hold the 80286 addresses when the 8207 cannot respond immediately. The 8207 will require a separate address latch, with the ALE input replaced with LEN. LEN optimizes the address setup and hold times for the 8207.

LEN goes from high to low when a valid 8207 command is recognized, which latches the 80286 address. This transition of LEN is independent of a memory cycle starting. The low to high transition will occur in the middle of a memory cycle so that the next address will be admitted and subsequently latched.

If Port B is to interface to an 80286 with the synchronous status interface, then LEN must be created using external logic. Figure 11 shows the equivalent 8207 circuit for Port B.

LOCK

The LOCK input allows each port uninterrupted access to memory. It does this by not permitting MUX to switch. It is not intended as a means to improve throughput of one of the ports. To do so is at the designer's risk[1]. Obviously, LOCK is only used in dual port systems. The 8207 interprets LOCK as originating from the port that MUX is indicating.

(1) The 8207 will not malfunction if this is done. This is a system level concern. For example, a time dependent process may fail if the other port holds LOCK active, preventing its access of memory and relinquishing the bus.

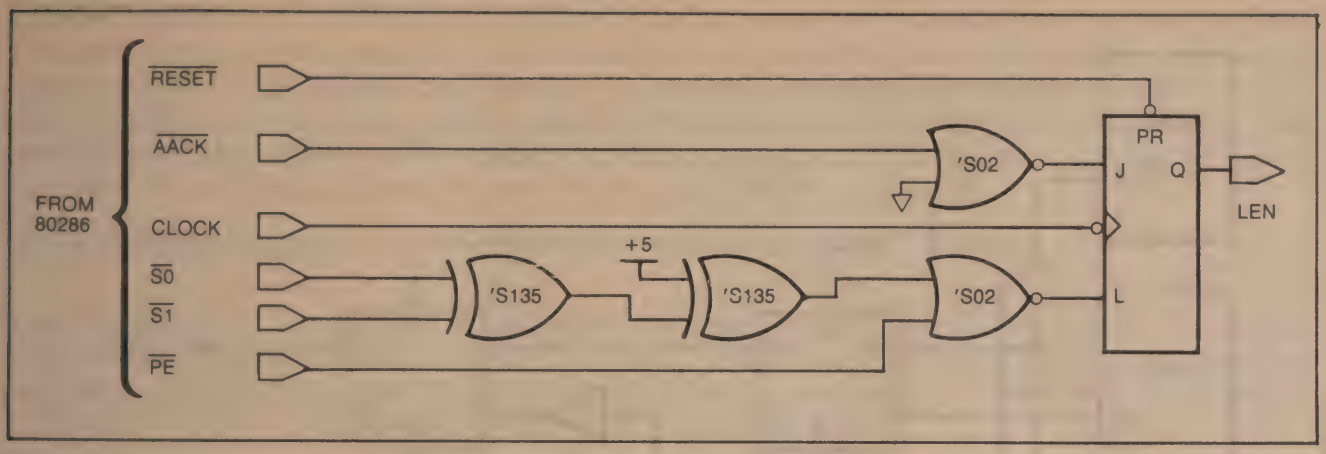


Figure 11. Port B LEN Circuit

LOCK from the 8086 may be connected directly to the 8207 or to the multiplexing logic. The 8207 requires additional logic when interfaced to an 80286. Figure 12 shows both the synchronous and asynchronous circuitry.

For 16 MHz operation, the 8207 ignores the LOCK input during the clock period that MUX switched. During 8 MHz operation, the 8207 will see LOCK as being active during the clock period when MUX switches.

The LOCK issued in Multibus bus systems may not be compatible with the 8207. The 8207 references LOCK from the beginning of a cycle, while Multibus references LOCK from the end of a cycle. The

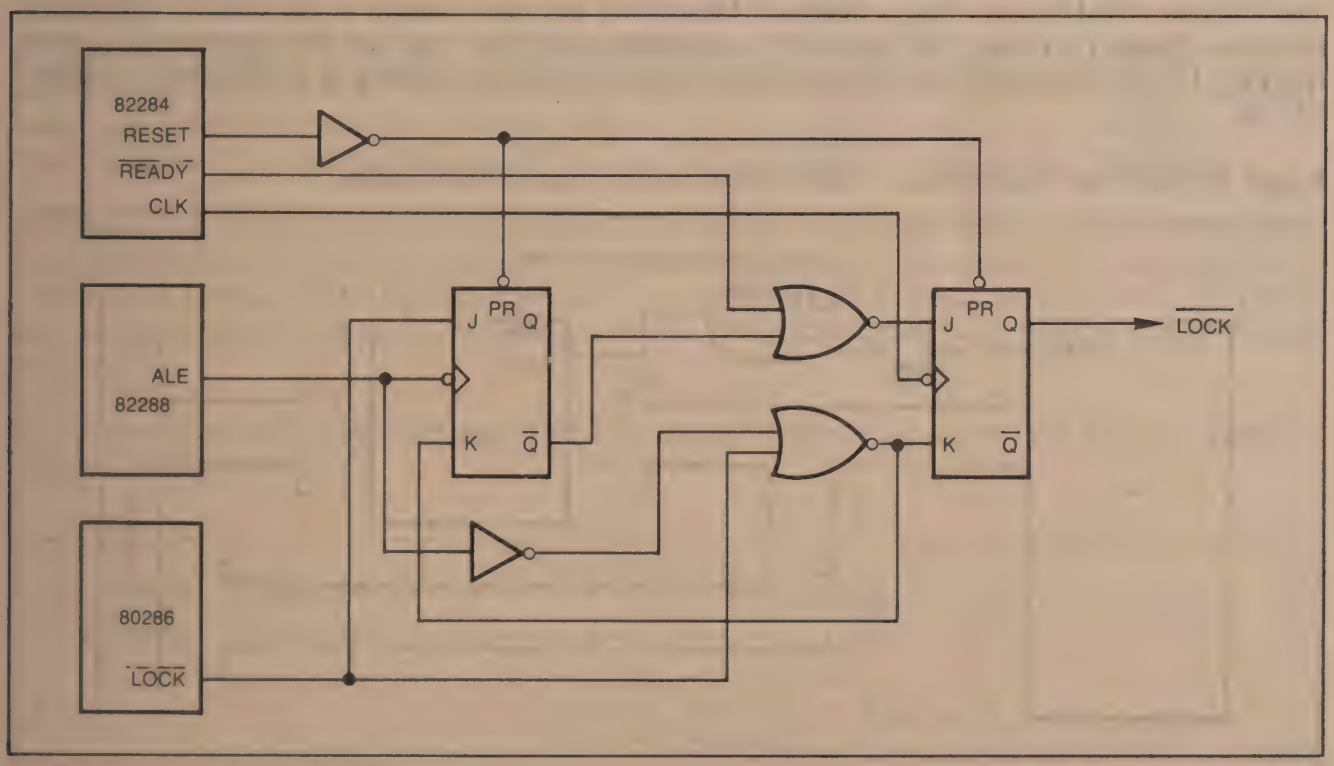


Figure 12a. Synchronous interface

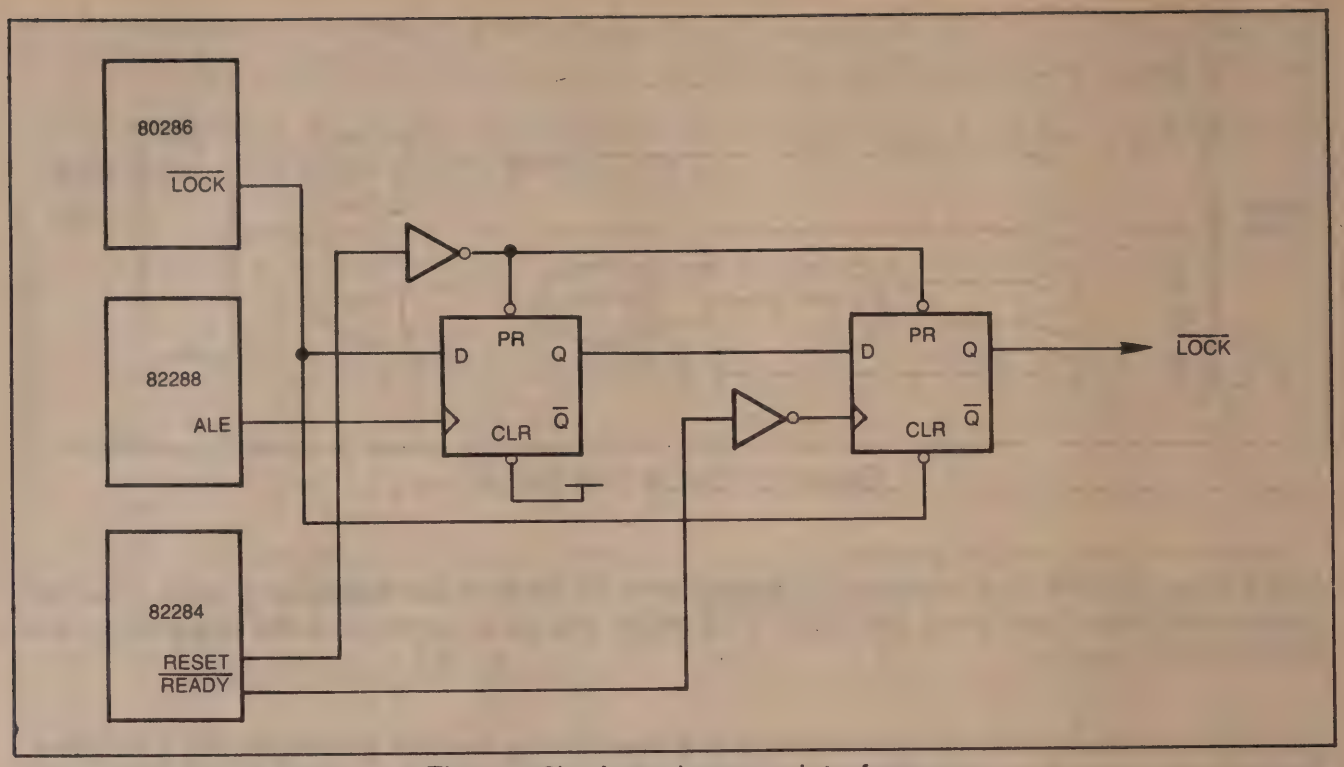


Figure 12b. Asynchronous interface

Multibus LOCK can be used if it meets the 8207 requirements. If the LOCK timing cannot be guaranteed, then additional logic is necessary. The logic would issue LOCK whenever a Multibus command is recognized. The drawback to this is that MUX cannot switch during the RAM cycle. This would delay the other port's memory access by one or two clocks.

DEADLOCK

The designer should ensure that a deadlock hazard has not been created in the design. The simple interfaces shown previously will not create a deadlock condition when the 8207 controls all system memory. If LOCK is issued by both ports, then the above logic would need to be modified to remove LOCK.

Figure 13 shows an illustration of the problem with a single LOCK input.

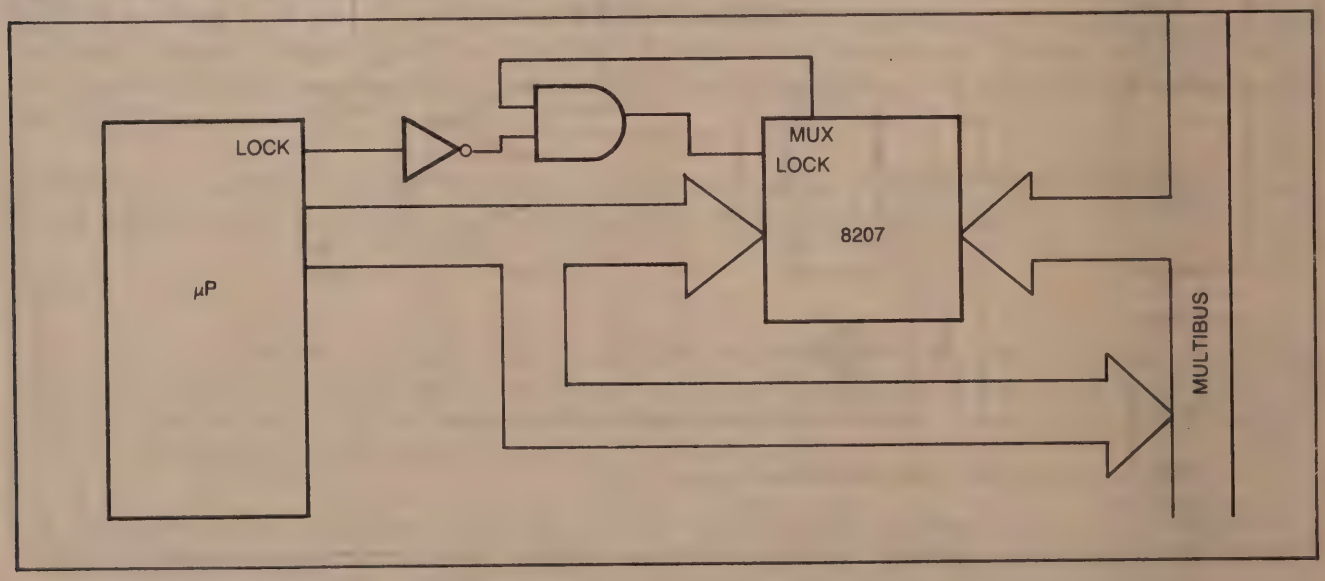


Figure 13. Single LOCK Input Circuit

Suppose the 8207 starts a locked string transfer for the processor. The Multibus bus port requests a memory cycle but must wait for the processor to remove LOCK. But the processor must access Multibus as part of the locked string transfer. We now have a deadlock. The solution is to force LOCK inactive whenever an access is made to non-8207 memory by the processor. By doing this we have now violated the purpose of LOCK, since the Multibus port could change data. Another solution is to ensure that locked data does not exist in physically separate memory.

8207 Acknowledge's

The 8207 in non-ECC mode has two active acknowledge's per port, $\overline{AACK}$ and $\overline{XACK}$. The $\overline{AACK}$ output is configured into either an "early" or "late" $\overline{AACK}$ based on the SA, SB bits in the program data word. In ECC systems there is one Acknowledge per port, and it is configured to any one of the three ($\overline{EAACK}$, $\overline{LAACK}$ or $\overline{XACK}$) by the programming bits.

The $\overline{AACK}$ pin is optimized for either the 80286 or the 8086, based upon the CFS programming bit (fast = 80286; slow = 8086). $\overline{XACK}$ conforms to the Multibus bus specification. $\overline{XACK}$ requires a tri-state buffer and must not drive the bus directly.

In synchronous systems, $\overline{XACK}$ will not go active if the memory command is removed prior to the clock period that issues $\overline{XACK}$. In asynchronous systems, the $\overline{AACK}$ pin can also serve as an advanced RAM cycle timing indicator.

Data out, in synchronous systems, should not have to be latched. The 8207 was designed to meet the data setup and hold times of Intel processors, the 8086 family, and the 80286. In asynchronous systems, the 8207 will remove data before the processor recognizes the Acknowledge ($\overline{LAACK}$ or $\overline{XACK}$). In these systems, the data should be latched with transparent type latches (Intel 8282/8283).

Output Data Control

Non-ECC

In single port systems, Intel processors supply the necessary timing signals to control the input or output of data to the RAMs. These control signals are $\overline{DEN}$ and DT/$\overline{R}$. Refer to the microprocessor handbook for their explanation. If these signals are not available, then PSEN and $\overline{DBM}$ provide the same function. They can be used directly to control the 8286/8287 bus drivers of the 8207.

Because of the single set of data in/out pins of the RAMs, data must be multiplexed between the two ports in dual port systems. The 8207 provides two outputs for contention-free switching. PSEL operates the same as the MUX output, in that a high selects Port A and a low selects Port B. PSEN acts to enable the selected port. The timing is shown in the 8207 Data Sheet, Port *Switching Timing* section.

The easiest means of using PSEL and PSEN is shown in Figure 14. At no time will both ports be enabled simultaneously.

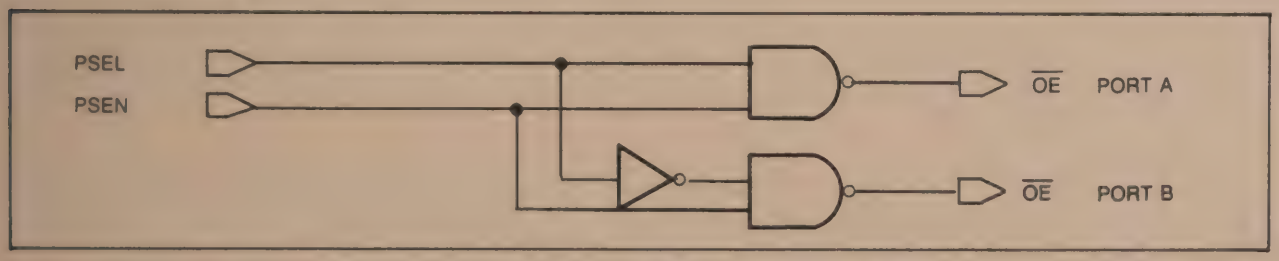


Figure 14. PSEL and PSEN Interface Circuit

Data Bus - Single Port

Recall that the 8207 always performs a late write cycle and that this requires separate data in and out buses. One option for the data bus is shown in Figure 3 of the 8207 Data Sheet. It requires separate data in and out traces on the processor board.

The second option is to keep the processor's combined data, bus but separate the data at the 8207 RAM. This is shown in Figure 15.

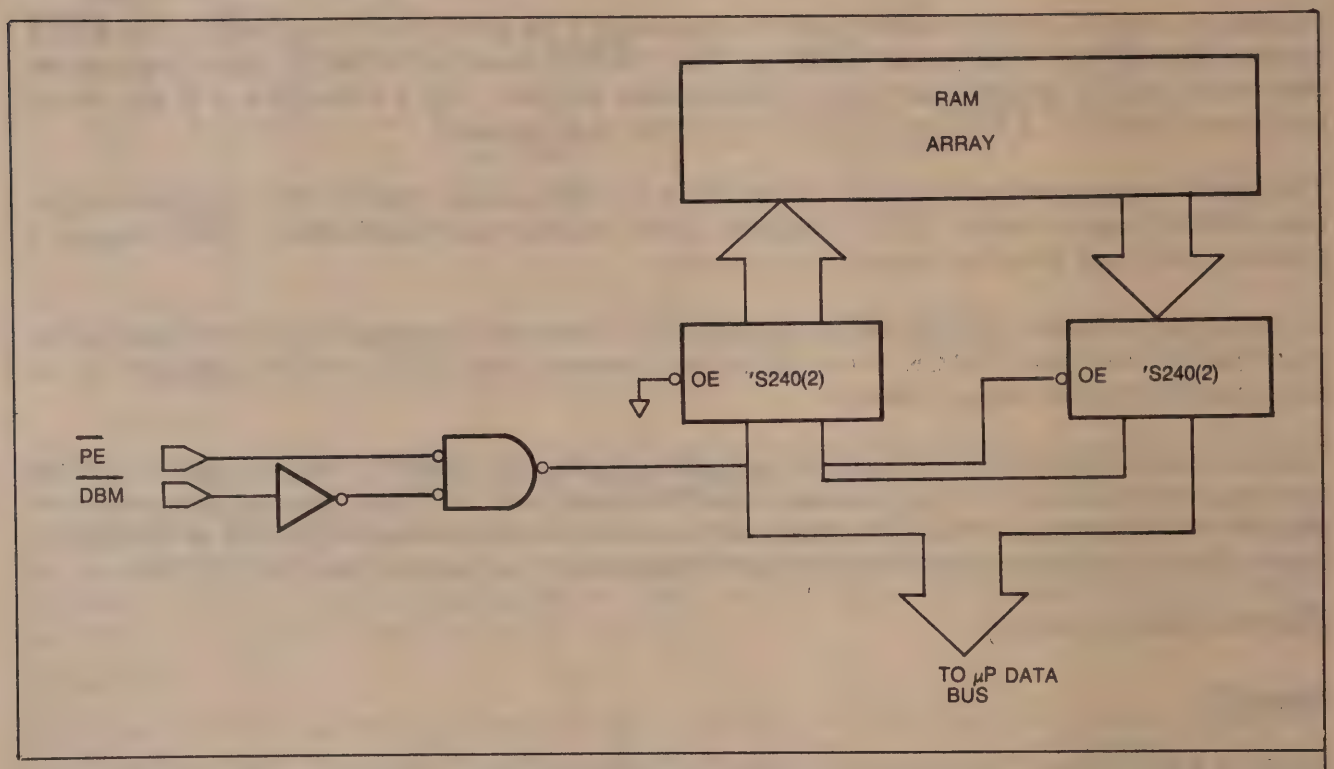


Figure 15. Data Bus Circuit

Data Bus - Dual Port

Non-ECC

The multiplexed data of the 8207 RAM must be kept isolated so that an access by one port does not affect another port. Figure 16 illustrates the control logic.

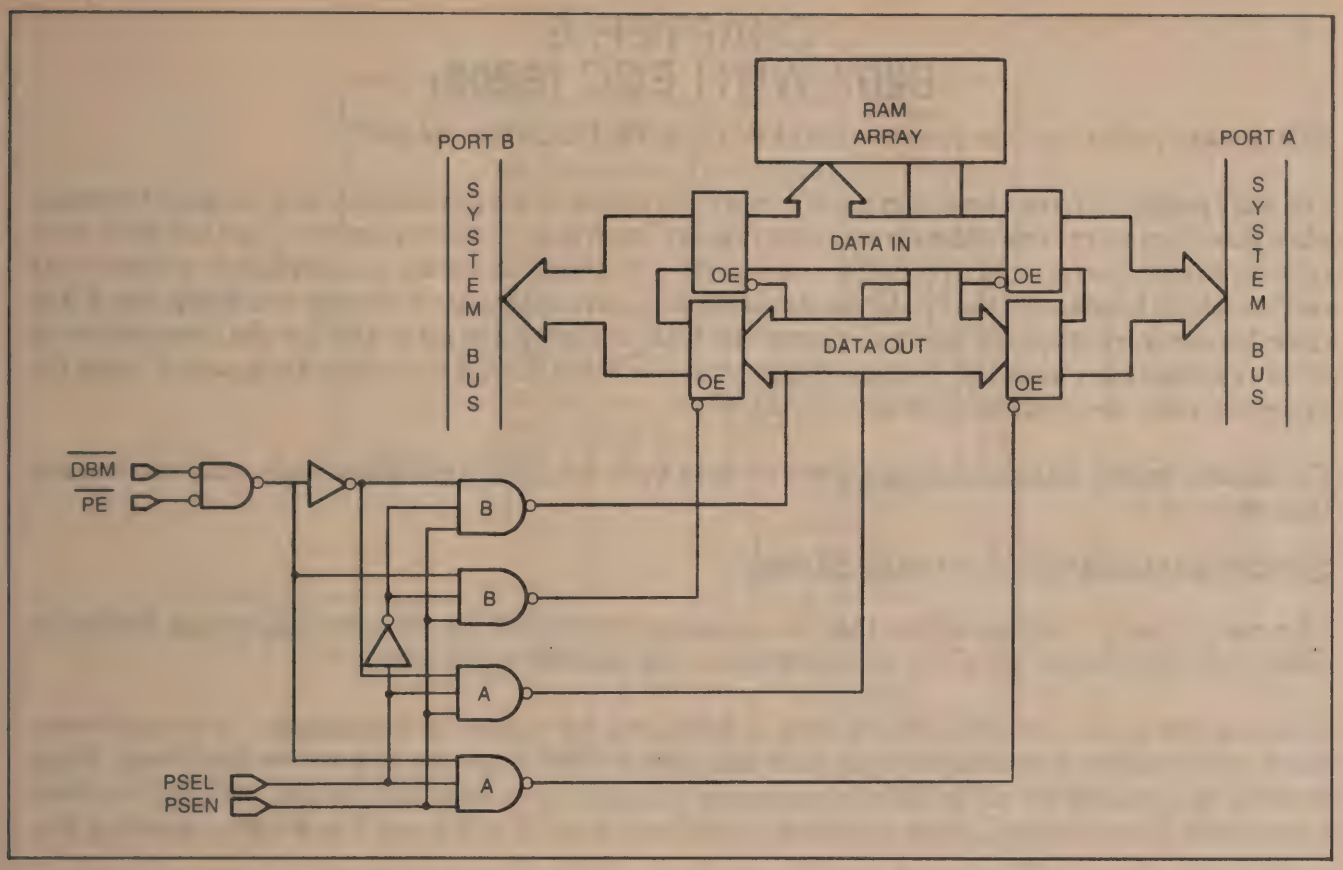


Figure 16. Dual Port Data Bus Control Circuitry

CHAPTER 5
8207 WITH ECC (8206)

This section points out the proper control of the 8206 EDCU by the 8207.

The 8207 performs error correction during read and refresh cycles (scrubbing), and initializes memory after power up to prevent false errors from causing interrupts to the processor. Since the 8207 must refresh RAM, performing scrubbing during refresh allows it to be accomplished without any additional performance penalty. Upon detection of a correctable error during scrubbing, the RAM refresh cycle is lengthened slightly to permit the 8206 to correct the error and for the corrected word to be rewritten into memory. Uncorrectable errors detected during scrubbing are ignored, since the processor may never access that memory location.

Correctable errors detected during a memory read cycle are corrected immediately and written back into memory.

Synchronous/Asynchronous Buses

The many types of configurations that are supported by the 8207/8206 combination can be broken down into two classes: ECC for synchronous or for asynchronous buses.

In synchronous bus systems, performance is optimized for processor throughput. In asynchronous buses, performance is optimized to get valid data onto the bus as quickly as possible (Multibus). While possible to optimize the 8207/8206 for processor throughput in Multibus systems, it is not Multibus compatible. The performance optimization is selected via the XA/XB and SA/SB programming bits.

When optimized for processor throughput, an advanced acknowledge ($\overline{AACK}$ - early or late) is issued at some point (based on the type of processor) so that data will be valid when the processor needs it.

When optimized for quick data access, an $\overline{XACK}$ is issued as soon as valid data is known to exist. If the data was invalid (based on the ERROR flag), then the $\overline{XACK}$ is delayed until the 8206 corrects the data and the data is on the bus.

The first example is known as "correct always" mode. The 8206 $\overline{CRCT}$ pin is tied to ground and the 8206 requires time to do the correction. Figure 17 shows this implementation. The quick data access method is known as "correct on error." The CRCT pin is tied to the R/$\overline{W}$ output of the 8207. When CRCT is high, the 8206 does not do correction, but still checks the data. This delay is typically half of the first. If an error happens, the cycle becomes a RMW and $\overline{XACK}$ is delayed slightly so that data can be corrected.

The correct on error mode is of no real benefit to non-Multibus users. The earliest acknowlege (EAACK) is delayed by one clock to allow for the delays through the 8206. This imposes a 1 wait state delay.

Byte Marks

The only real difference to the 8207 system when adding the 8206 is the treatment of byte writes. Because the encoded check bits apply only to a whole word (including check bits), byte writes must not be permitted at the RAM. Instead, the altering of byte data is done at the 8206. The byte marks previously sent to RAM are now sent to the 8206. These byte marks must also qualify the output enables of the data drivers.

The $\overline{DBM}$ output of the 8207 is meant to be nanded with the processors byte marks. This output is activated only on reads or refreshes. On write cycles, this output stays high which would force the 8206 byte mark input low. When low, the internal 8206 data out buffers are tristated so that new data may be gated into the device.

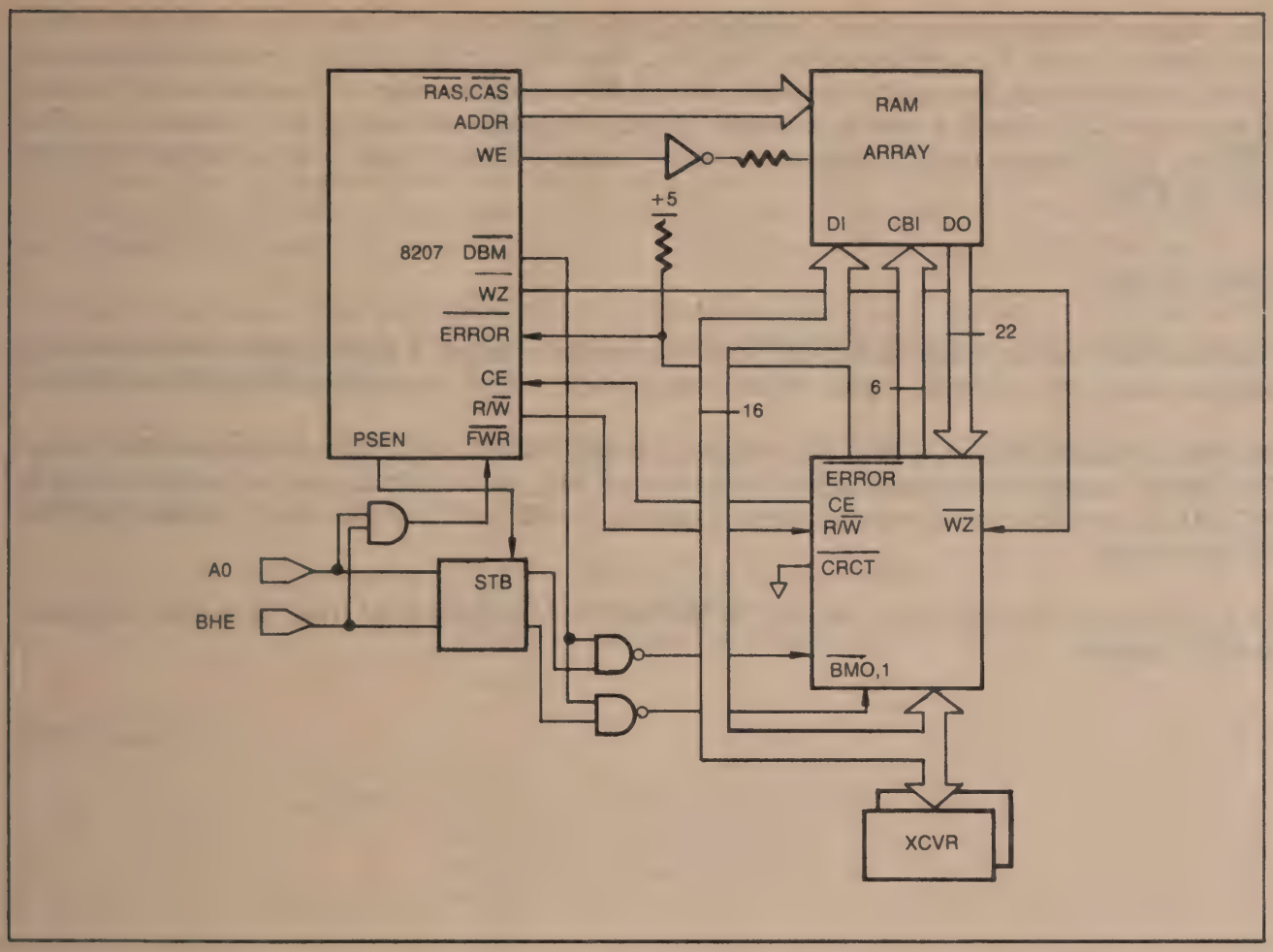


Figure 17. 8206 Interface to the 8207

Read Modify Writes - ECC

A RMW cycle occurs whenever a processor wants to do byte writes or when the 8207 has detected an error during read or refresh (scrubbing) cycles. A byte write is detected by the $\overline{FWR}$ input to the 8207 and is based on the processor supplied byte marks.

At the start of a RMW cycle, $\overline{DBM}$ stays high, which, when qualified with the byte marks, will enable the data out buffer of the 8206 for the unmodified byte, and tristates the buffer for the new byte; R/$\overline{W}$ is high, which tells the 8206 to do error detection and correcting (if $\overline{CRCT}$ is low). The 8206 can latch data and check bits from the RAM via the STB input, but the 8207 does not use this feature. Instead, the 8207 keeps $\overline{CAS}$ active the entire length of the RMW cycle to hold data at the 8206. The new byte data from the processor goes to the 8206 and to the RAM. The 8207 would have corrected any errors just read, so the old and new bytes of data, plus their check bits, are available at the RAM, and the 8207 generates a write pulse. The data driver for the unmodified byte must not have been enabled, otherwise erroneous data would be written to RAM and possibly made valid (if it was stable) by the 8206.

Data Buffer Control - ECC

The control of the data buffers is essentially the same as in non-ECC systems, with a few exceptions.

The processor's byte marks must now qualify the output enable logic. The reason was described earlier in the RMW section. This applies to both single and dual port configurations. A refresh cycle outputs all the control signals that a read cycle will, except for an acknowledge. If complete buffer control is left to the 8207, then it would occasionally (during refreshes) put data on the processor bus. The $\overline{DEN}$ and DT/$\overline{R}$ signals must be qualified by the $\overline{PE}$ input. $\overline{PE}$ would have to be latched for the entire cycle by PSEN.

Test Modes

Neither of the two test modes of the 8207 are to be used in a design. Both test modes reset the refresh address counter to a specific value, which interrupts the refresh sequence and causes loss of data.

In error corrected systems, a reset pulse causes the 8207/8206 to write over the entire RAM array. Test Mode 2 appears to bypass the prewrite sequence. But, the refresh counter is reset to a value of 1F7 (H). So, besides interrupting the refresh sequence, the 8207 still prewrites the 8 locations specified by the counter.

To not overwrite the RAM data, the 8207 RESET will have to be isolated from the system reset logic in ECC systems.

APPENDIX I

8207/8208 Performance

The following performance charts were based upon Figure 3 in the 8207 Data Sheet, and apply to the 8208 as well. All RAM access delays are based upon Intel dynamic RAMs. The charts show the performance of a single cycle with no precharge, refresh, port switching, or arbitration delays.

The read access calculations are: the margin between the 8207 starting a memory cycle to data valid at the processor - 8207 RAS or CAS from clock delay - DRAM RAS or CAS access - 8286 propagation delay - processor setup.

Assume the RAS/CAS drivers are loaded with 150 pf, and the 8286 is driving a 300 pf data bus.

80286 (example)

RAS Access: 3TCLCL - 8207 TCLRSL - 2118 tRAC - 8286 TIVOV - 80286 t8
= (3)62.5 - 35 max - 100 max - 22 - 10
= 20 ns

80186 (example)

CAS Access: 2 TCLCL - 8207 TCLCSL - 2164A tCAC - 8286 TIVOV - 80186 TDVCL
= (2)125 - 115 max - 85 max - 22 - 20
= 8 ns

8207 Performance (EDC synchronous status interface)

Table 5a. Wait States for Different μP and RAM Combinations

Wait states at full CPU speed		RAM speed			
CPU	Freq	100 ns	120 ns	150 ns	200 ns
80286	8 MHz	1-RD, WR 3-Byte WR CO (3)	1-RD, WR 3-Byte WR CO	2-Read 1-Write 3-Byte WR C2	Not (1) compatible with RAM parameters
80186, 8086/88-2	8 MHz	1-RD, WR 3-Byte WR C4	1-RD,WR 3-Byte WR C4	1-RD,WR 3-Byte WR C4	
8086/88	5 MHz	1 C6	1 C6	1 C6	1-RD, WR 3-Byte WR C4

8207 Performance (EDC synchronous status interface)

Table 5b. μP Clock Frequency for Differenc μP and RAM Combinations

Maximum frequency for one wait-state (4)		RAM speed			
CPU	Freq	100 ns	120 ns	150 ns	200 ns
80286	8 MHz	FULL SPEED		7.3 MHz C0	6 MHz C0
80186, 8086/88-2	8 MHz				7 MHz C4
8086/88	5 MHz				

8207 Performance (Non-EDC synchronous status interface)

Table 6a. Wait States for Different μP and RAM Combinations

Wait states at full CPU speed		RAM speed			
CPU	Freq	100 ns	120 ns	150 ns	200 ns
80286	8 MHz	0 CO(3)	1-Read 0-Write C1	1-Read 0-Write C1	Not(1) compatible with RAM parameters
80186, 8086/88-2	8 MHz	0 C3	0 C3	0(2) C3	
8086/88	5 MHz	0 C3	0 C3	0 C3	0 C3

Table 6b. μP Clock Frequency for Different μP and RAM Combinations

Maximum frequency for no wait-state (4)		RAM speed			
CPU	Freq	100 ns	120 ns	150 ns	200 ns
80286	8 MHz	FULL SPEED	7 MHz	6 MHz	5.3 MHz
80186, 8086/88-2	8 MHz		FULL SPEED		7 MHz
8086/88	5 MHz				

(1) The 2164A tRAH parameter is not satisfied.

(2) 150 ns 64K DRAMs with tCAC = 100 ns won't run with 0 wait-states, because they have a longer CAS access time than the 2164A-15 (tCAC = 85 ns).

(3) Numbers in lower right corners are the programmed configurations of the 8207.

(4) To meet read access time.

8207 Performance (multibus interface)

This is an *asynchronous, command interface*. Worst case data and transfer acknowledge (XACK#) delays. Including synchronization and data buffer delays, are:

Table 7a. Non-EDC system

	RAM speed			
	100 ns	**120 ns**	**150 ns**	**200 ns**
Data access time	289ns	299ns	322ns	380ns
XACK# access time	333ns			450ns

Table 7b. EDC system

	RAM speed			
	100 ns	**120 ns**	**150 ns**	**200 ns**
Data access time (read)	359ns (324 ns)[1]	369ns (334 ns)	392ns (357 ns)	450ns (415 ns)
XACK# access time	400 ns-RD, WR 588 ns-Byte Write			520 ns-RD, WR 806 ns-Byte WR

(1) Numbers in parentheses are for when 8206 is in check-only mode (8206 doesn't do error correction until after an error is detected.

intel

APPLICATION NOTE

AP-97A

April 1982

Interfacing Dynamic RAM to iAPX 86, 88 Systems Using the Intel 8202A and 8203

Brad May
Peripheral Component
Applications Engineering

INTRODUCTION

The designer of a microprocessor-based system has two basic types of devices available to implement a random access read/write memory — static or dynamic RAM. Dynamic RAMs offer many advantages. First, dynamic RAMs have four times the density (number of bits per device) of static RAMs, and are packaged in a 16-pin DIP package, as opposed to the 20-pin or larger DIPs used by static RAMs; this allows four times as many bytes of memory to be put on a board, or alternatively, a given amount of memory takes much less board space. Second, the cost per bit of dynamic RAMs is roughly one-fourth that of statics. Third, static RAMs use about one-sixth the power of static RAMs, so power supplies may be smaller and less expensive. These advantages are summarized in Table 1.

On the other hand, dynamic RAMS require complex support functions which static RAMs don't, including

- address multiplexing
- timing of addresses and control strobes
- refreshing, to prevent loss of data
- arbitration, to decide when refresh cycles will be performed.

Table 1. Comparison of Intel Static and Dynamic RAMs Introduced during 1981

	2164-15 (Dynamic)	2167-70 (Static)
Density (No. of bits)	64K	16K
No. of pins	16	20
Access time (ns)	150	70
Cycle time (ns)	300	70
Active power (ma)	60	125
Standby power (ma)	5	40
Approx. cost per bit (millicents/bit)	45	250

In addition, dynamic RAMs may not always be able to transfer data as fast as high-performance microprocessors require; wait states must be generated in this case. The circuitry required to perform these functions takes up board space, costs money, and consumes power, and so detracts from the advantages that make dynamic RAMs so appealing. Obviously, the amount of support circuitry should be minimized.

The Intel 8202A and 8203 are LSI dynamic RAM controller components. Either of these 40-pin devices alone does all of the support functions required by dynamic RAMs. This results in a minimum of board space, cost, and power consumption, allowing maximum advantage from the use of dynamic RAMs.

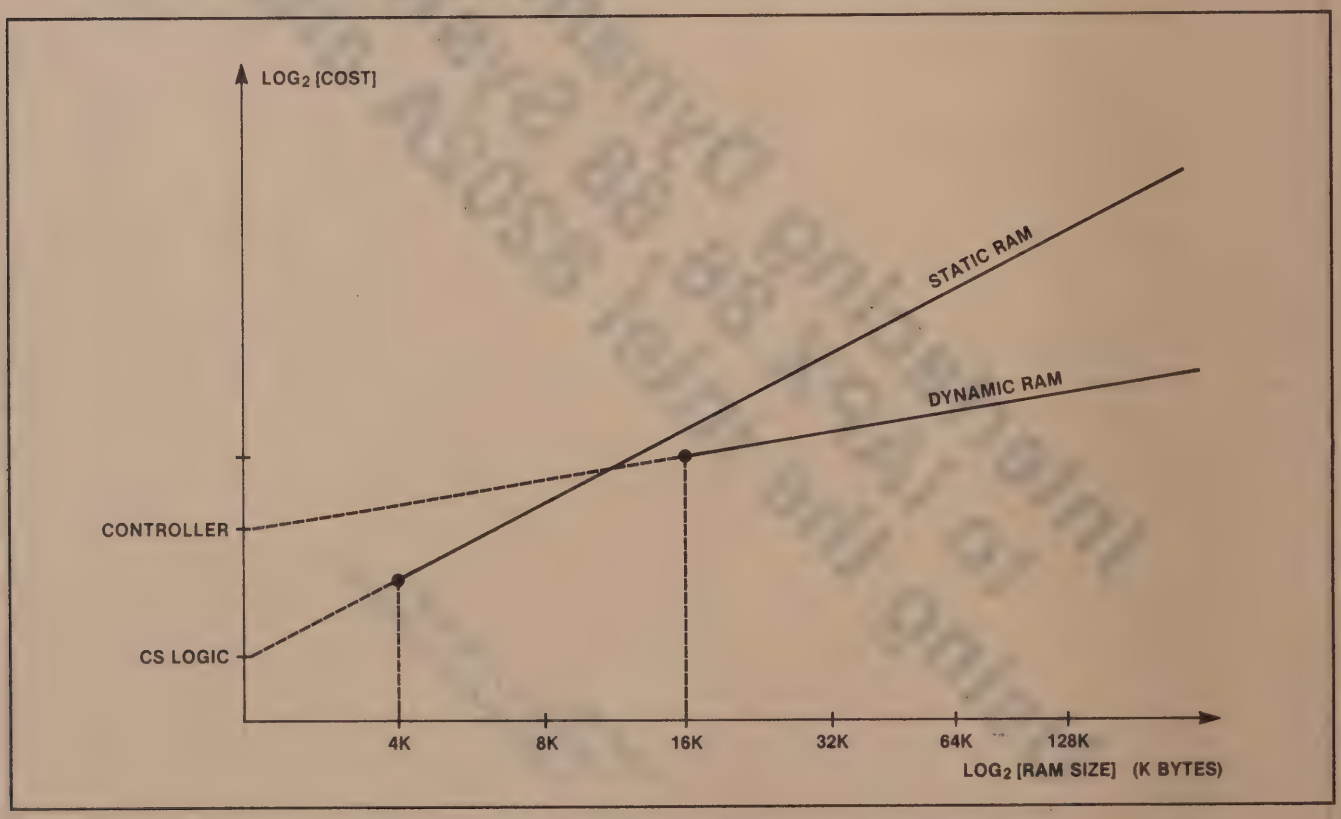


Figure 1. Implemented Cost of Static vs. Dynamic RAM

Figure 1 shows the relative cost of static and dynamic RAM, including support circuitry, as a function of memory size, using the Intel 8202A or 8203. For any memory larger than 16KBytes, the dynamic RAM is less expensive. Since the cost of the dynamic RAM controller is relatively independent of memory size, the cost advantge for dynamic RAM increases with increasing memory size.

This Application Note will describe the techniques of interfacing a dynamic RAM memory to an iAPX-86 or iAPX-88 system using either the 8202A or 8203 dynamic RAM controller. Various configurations of the 8086 and 8088 microprocessors, and those timings which they satisfy, are described. The Note concludes with examples of particular system implementations.

DYNAMIC RAMS

This section gives a brief introduction to the interfacing requirements for Dynamic RAMs. Later sections will describe the operation of the Intel 8202A and 8203 Dynamic RAM Controllers.

Device Description

The pinout of two popular families of dynamic RAMs, the Intel 2118 and 2164A, are shown in Figure 2. The 2118 is a 16,384 word by 1-bit dynamic MOS RAM. The 2164 is a 65,536 word by 1-bit dynamic MOS RAM. Both parts operate from a single +5v supply with a ± 10% tolerance, and both use the industry standard 16-lead pinout.

The two parts are pinout-compatible with the exception of the 2164 having one extra address input (A_7, pin 9); this pin is a no-connect in the 2118. Both parts are also compatible with the next generation of 256K dynamic RAMs (262,144 word by 1-bit), which will use pin 1 (presently a no-connect on both the 2118 and 2164A) for the required one extra address input (A_8). This makes it possible to use a single printed circuit board layout with any of these three types of RAM.

Addressing

Each bit of a dynamic RAM is individually addressable. Thus, a 2164A, which contains 2^{16} (or 65,536) bits of information, requires 16-bit addresses; similarly, the 2118, which contains 2^{14} (or 16,384) bits, requires 14-bit addresses.

In order to reduce the number of address pins required (and thus reduce device cost), dynamic RAMs time-multiplex addresses in two halves over the same pins. Thus a 2164A needs only 8 address pins to receive 16-bit addresses, and the 2118 needs only 7 for its 14-bit addresses. The first address is called the *row* address, and the second is called the *column* address. The row address is latched internal to the RAM by the falling edge of the $\overline{RAS}$ (*R*ow *A*ddress *S*trobe) control input; the column address is latched by the falling edge of the $\overline{CAS}$ (*C*olumn *A*ddress *S*trobe) control input. This operation is illustrated in Figure 3.

Dynamic RAMS may be visuallized as a two-dimensional array of single-bit storage cells arranged across the surface of the RAM's die. In the case of the 2164A, this array would consist of 2^8 (or 256) rows and 2^8 (or 256) columns, for a total of 2^{16} (or 65,526) total bit cells (Figure 4). This is the source of the "row address" and "column address" terminology. Bear in mind that any given RAM may not be physically implemented as described here; for instance, the 2164A actually contains four arrays, each one 2^7 rows by 2^7 columns.

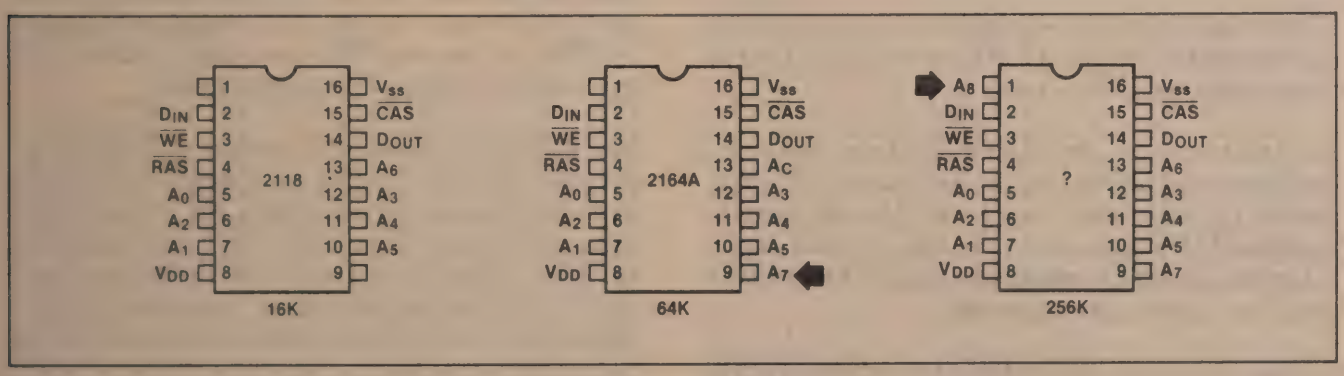


Figure 2. Dynamic RAM Pinout Compatibility

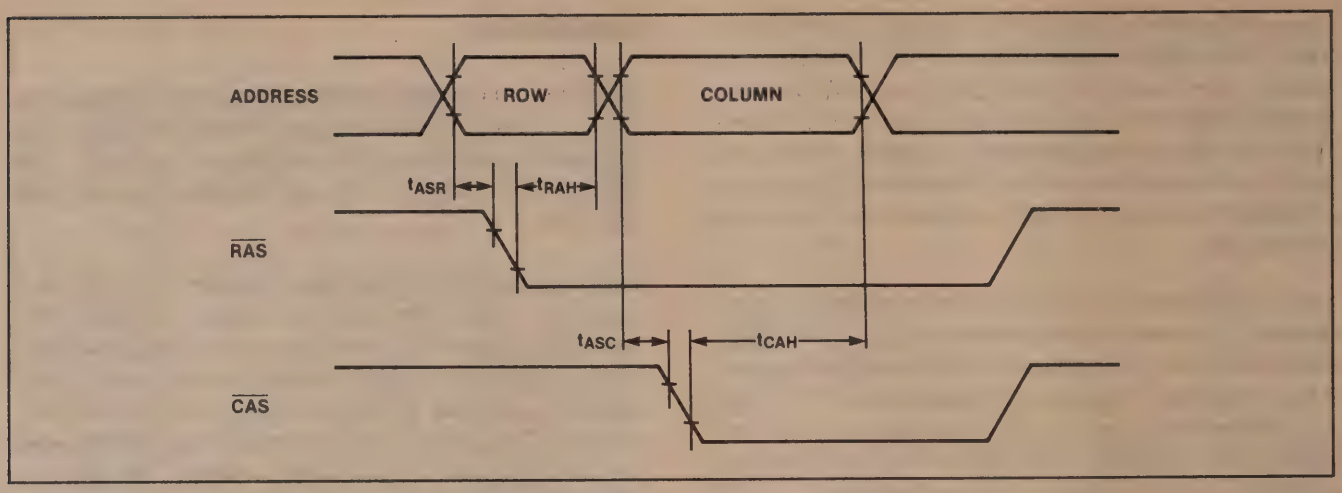


Figure 3. Dynamic RAM Addressing

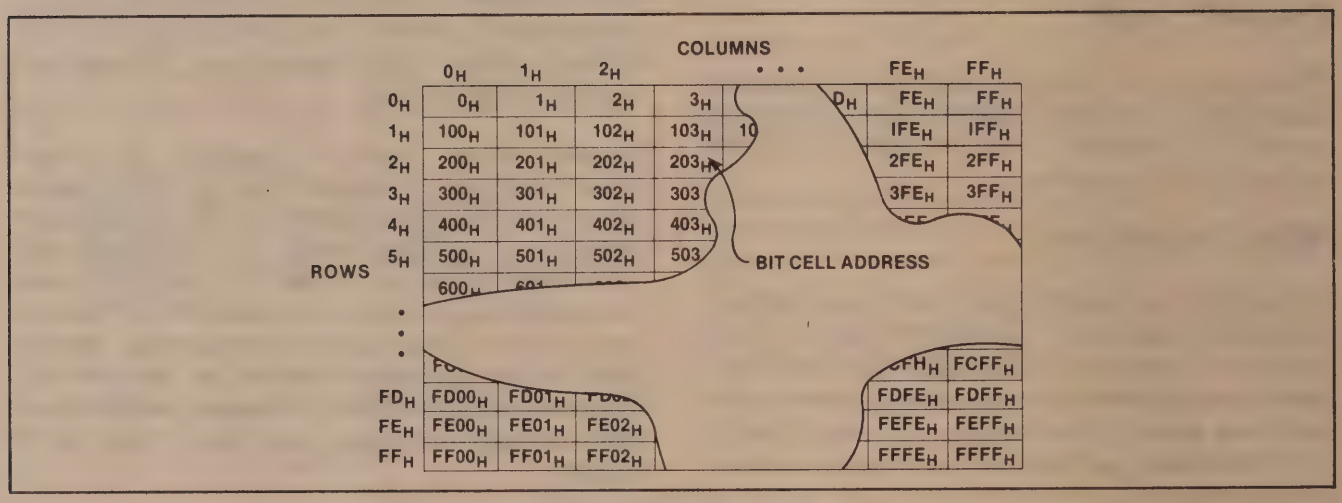


Figure 4. Bit Cell "Array"

Memory Cycles

In this Application Note, we will discuss three types of memory cycles — read, write, and $\overline{RAS}$-only refresh. Dymanic RAMs may perform other types of cycles as well; these are described in the dynamic RAM's data sheet.

Whether data is read or written during a memory cycle is determined by the RAM's $\overline{WE}$ control input. Data is written only when $\overline{WE}$ is active.

During a read cycle, the $\overline{CAS}$ input has a second function, other than latching the column address. $\overline{CAS}$ also enables the RAM data output (pin 14) when active, assuming $\overline{RAS}$ is also active. Otherwise, the data output is 3-stated. This allows multiple dynamic RAMs to have their data outputs tied in common.

During write cycles, data on the RAM data input pin is latched internally to the RAM by the falling edge of $\overline{CAS}$ or $\overline{WE}$, *whichever occurs last*. If $\overline{WE}$ goes active before $\overline{CAS}$ (the usual case, called an "early write"), write data is latched by the falling edge of $\overline{CAS}$. If $\overline{WE}$ goes active after $\overline{CAS}$ (called a "late write"), data is latched by the falling edge of $\overline{WE}$ (see Figure 5).

Late writes are useful in some systems where it is desired to start the memory cycle as quickly as possible, to maximize performance, but the CPU cannot get the write data to the dynamic RAMs quickly enough to be latched by $\overline{CAS}$. By delaying $\overline{WE}$, more time is allowed for write data to arrive at the dynamic RAMs.

Note that when "late write" is performed, $\overline{CAS}$ goes active while $\overline{WE}$ is still inactive; this indicates a read cycle, so the RAM enables its data output. So, if "late write" cycles are performed by a system, the RAM data inputs and data outputs must be electically isolated from each other to prevent contention. If no "late writes" are performed, the RAM data inputs and data outputs may be tied together at the RAM to reduce the number of board traces.

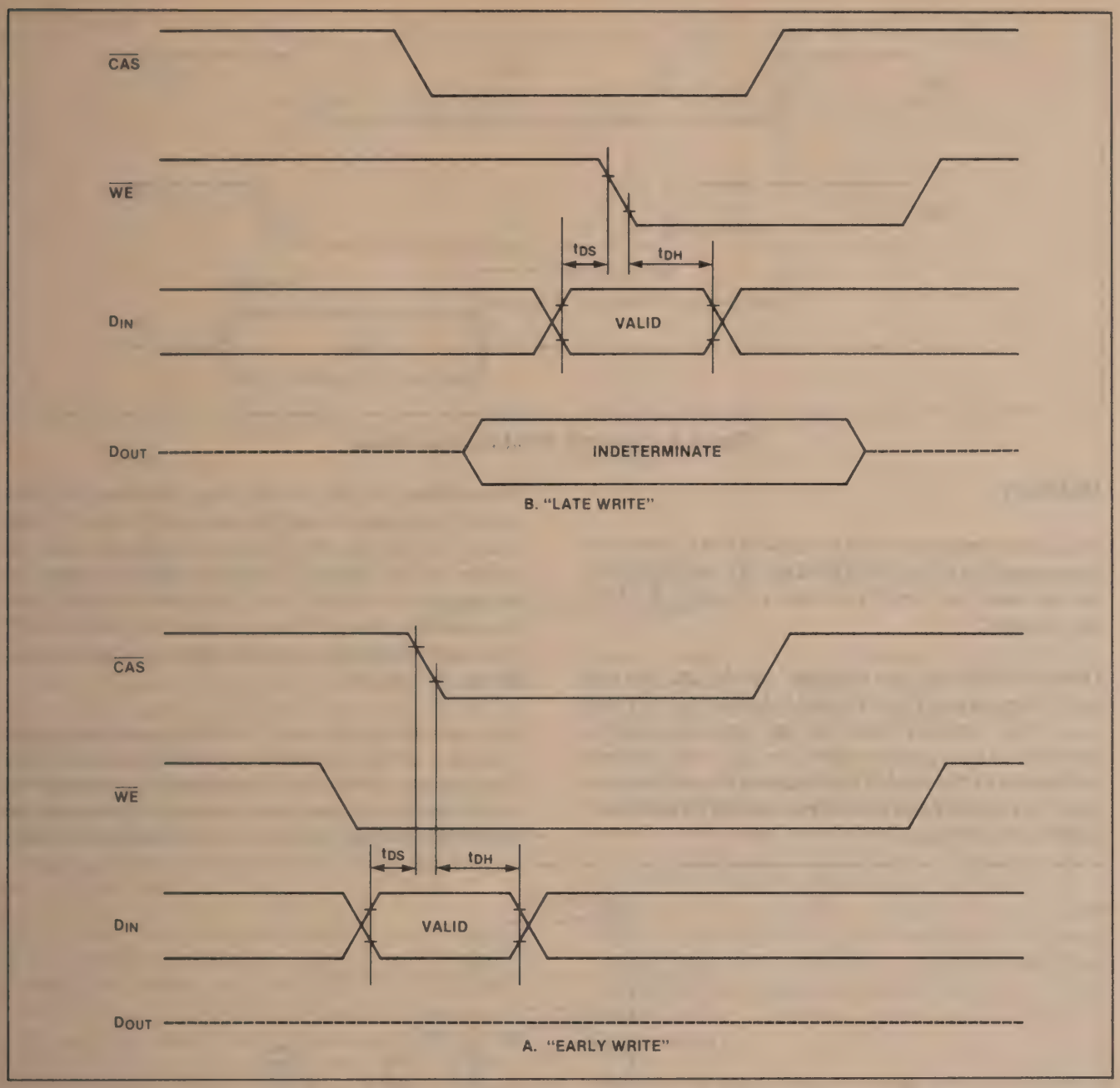


Figure 5. Dynamic RAM Write Cycles

Access Times

Each dynamic RAM has two different access times quoted for it — access time from $\overline{RAS}$ active (t_{RAC}) and access time from $\overline{CAS}$ active (t_{CAC}); these are illustrated in Figure 6. How do you know which to use? This depends on the timings of your RAM controller. First, the worst case delay from the memory read command active to $\overline{RAS}$ active (t_{CR}) and $\overline{CAS}$ active (t_{CC}) must be determined. Then the read data access time is the larger of the t_{CR}(Controller) + t_{RAC}(RAM) or t_{CC}(Controller) + t_{CAC}(RAM). An alternative way to determine whether to use t_{RAC} or t_{CAC} is to look at the dynamic RAM parameter for $\overline{RAS}$ active to $\overline{CAS}$ active delay, t_{RCD}. t_{RCD}*max* is a calculated value, and is shown on dynamic RAM data sheets as a reference point only. If the delay from $\overline{RAS}$ to $\overline{CAS}$ is less than or equal to t_{RCD}max, then t_{RAC} is the limiting access time parameter; if, on the other hand, the delay from $\overline{RAS}$ to $\overline{CAS}$ is greater than t_{RCD}max, then t_{CAC} is the limiting parameter. t_{RCD}max is not an operating limit, and this spec may be exceeded without affecting operation of the RAM. t_{RCD}*min*, on the other hand, is an operating limit, and the RAM will not operate properly if this spec is violated.

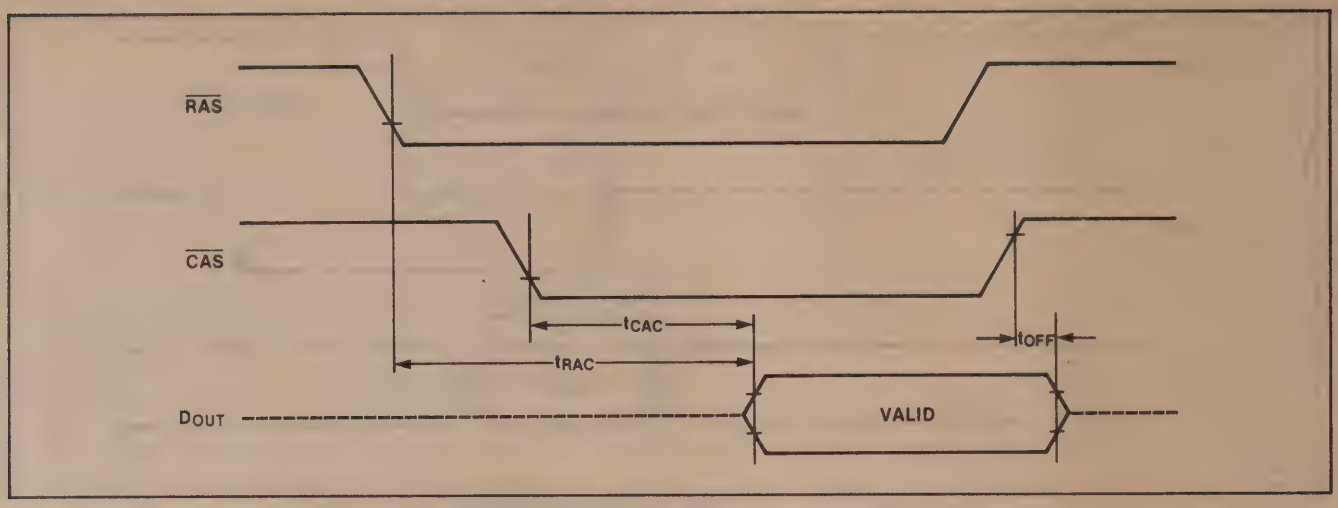


Figure 6. Dynamic RAM Access Times

Refresh

One unique requirement of dynamic RAMs is that they be *refreshed* in order to retain data. To see why this is so, we must look briefly at how a dynamic RAM is implemented.

Dynamic RAMs achieve their high density and low cost mostly because of the very simple bit-storage cell they use, which consists only of one transistor and a capacitor. The capacitor stores one bit as the presence (or absence) of charge. This capacitor is selectively accessed for reading and writing by enabling its associated transistor (see Figure 7).

Unfortunately, if left for very long, the charge will leak out of the capacitor, and the data will be lost. To prevent this, each bit-cell must be periodically read, the charge on the capacitor amplified, and the capacitor recharged to its initial state. The circuitry which does this amplification of charge is called a "sense amp". This must be done for every bit-cell every 2 ms or less to prevent loss of data.

Each column in a dynamic RAM has its own sense amp, so refresh can be performed on an entire row at a time. Thus, for the 2118, it is only necessary to refresh each of its 128 rows every 2 ms. Each row must be addressed via the RAM's address inputs to be refreshed. To simplify

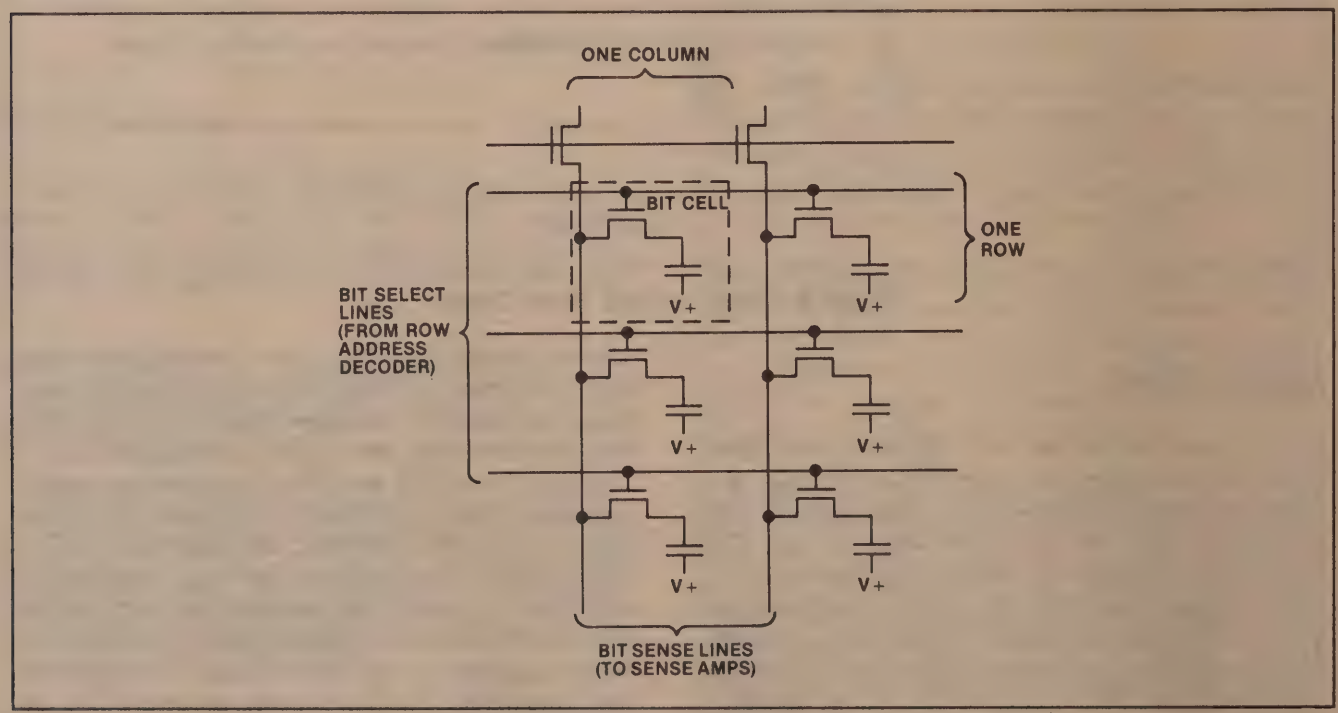


Figure 7. Dynamic RAM Cell

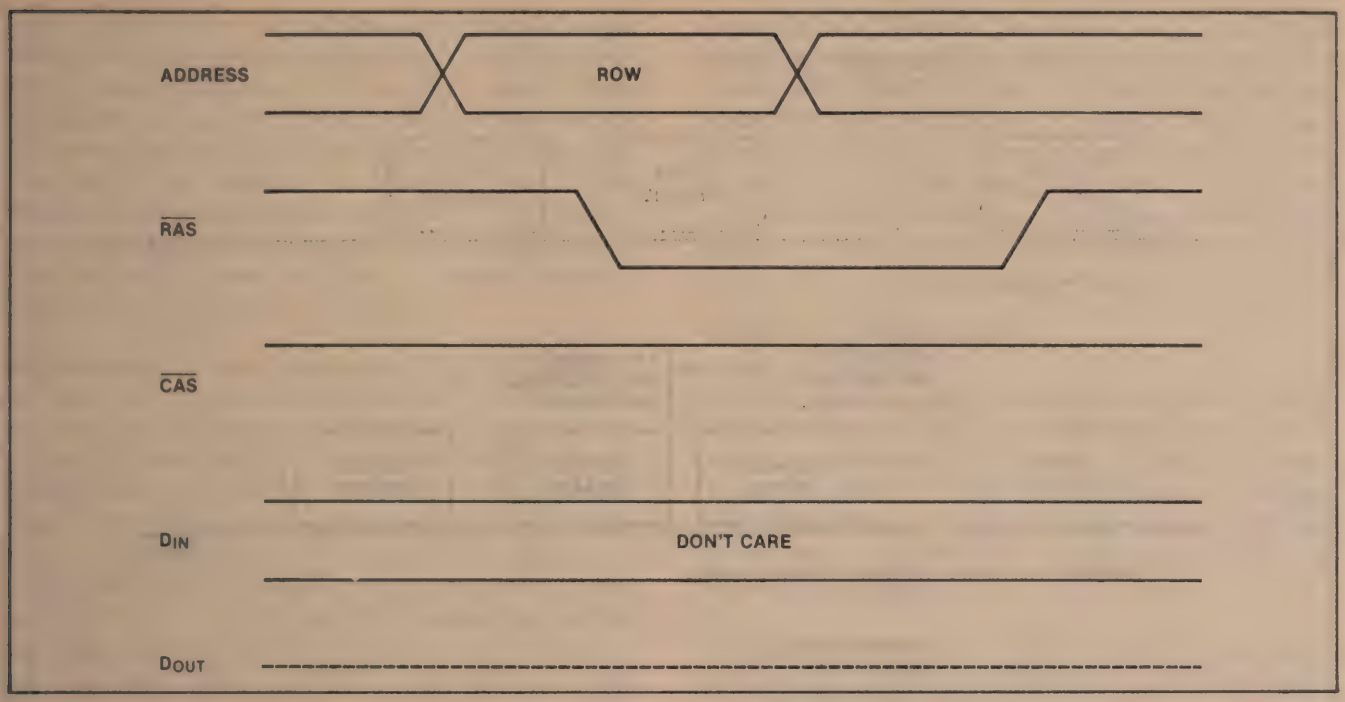


Figure 8. RAS-only Refresh

refresh, the 2164A is implemented in such a way that its refresh requirements are identical to the 2118; 128 rows every 2 ms. Some other 64K RAMs require 256 row refresh every 4 ms.

Refresh can be performed by a special cycle called a *RAS-only refresh*, shown in Figure 8. Only a row address is sent; that row is refreshed. No column address is sent, and no data is read or written during this cycle. Intel dynamic RAM controllers use this technique.

Any read, write, or read-modify-write cycle also refreshes the row addressed. This fact may be used to refresh the dynamic RAM without doing any special refresh cycles. Unfortunately, in general you cannot be sure that every row of every dynamic RAM in a system will be read from or written to every 2 ms, so refresh cannot be guaranteed by this method alone, except in special applications.

A third technique for refresh is called *hidden refresh*. This method is not popular in microprocessor systems, so it is not described here, but more information is available in the dynamic RAM's data sheet.

Three techniques for timing when refresh cycles are performed are in common use: burst refresh, distributed refresh, and transparent refresh.

Burst refresh means waiting almost 2 ms from the last time refresh was performed, then refreshing the entire memory with a "burst" of 128 refresh cycles. This method has the inherent disadvantage that during the time refresh is being performed (more than 40 microseconds for 128 rows) no read or write cycles can be performed. This severely limits the worst case response time to interrupts and makes this approach unsuitable for many systems.

As long as every row of the RAM is refreshed every 2 ms, the distribution of individual refresh cycles is unimportant. *Distributed refresh* takes advantage of this fact by performing a single refresh cycle every 2 ms/128, or about every 15 microseconds. In this way, the refresh requirements of the RAM are satisfied, but the longest time that read and write cycles are delayed because of refresh is minimized. Those few dynamic RAMs which use 256 row refresh allow 4 ms for the refresh to be completed, so the distributed refresh period is still 15 microseconds.

The third technique is called *transparent* (or "hidden" or "syncronous") *refresh*. This takes advantage of the fact that many microprocessors wait a fixed length of time after fetching the first opcode of an instruction to decode it. This time is necessary to determine what to do next (i.e. fetch more opcode bytes, fetch operands, operate on internal registers, etc.); this time may be longer than the time required for a RAM refresh cycle. If the status outputs of the CPU can be examined to determine which memory cycles are opcode fetches, a refresh cycle may be performed immediately afterward (Figure 9). In this way, refresh cycles will never interfere with read or write cycles, and so appear "transparent" to the microprocessor.

Transparent refresh has the disadvantage that if the microprocessor ever stops fetching opcodes for very

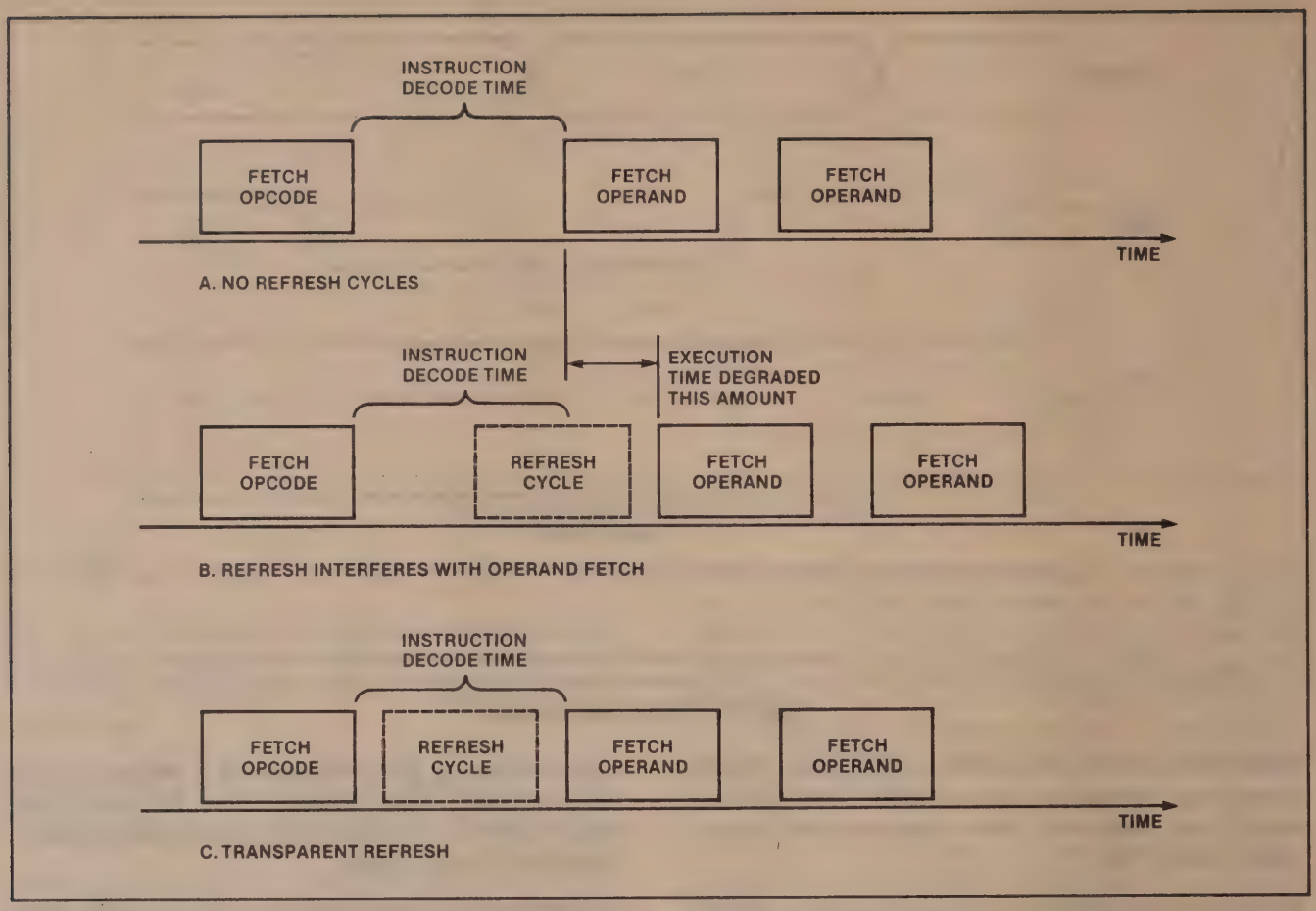


Figure 9. Transparent Refresh

long, due to a HOLD, extended DMA transfers, or when under hardware emulation, no refresh cycles will occur and RAM data will be lost. This puts restrictions on the system design. Also, high speed microprocessors do not allow sufficient time between opcode fetches and susequent bus cycles for a complete RAM refresh cycle to be performed, so they must wait for the refresh cycle to complete before they can do a subsequent bus cycle. These microprocessors cannot use transparent refresh to any advantage. Transparent refresh is useful for microprocessors like the Intel 8085 operating at low clock frequencies.

The 8086 and 8088, however, prefetch opcodes into a queue which is several bytes long. This prefetching is independent of the actual decoding and execution of the opcodes, and there is no time at which it can be guaranteed that the 8086 or 8088 will not request a memory cycle. So transparent refresh is not applicable to these microprocessors.

The 8202A and 8203 perform distributed and/or transparent refresh. Each device has an internal timer which automatically generates a distributed refresh cycle every 15.6 microseconds or less. In addition, an external refresh request input (REFRQ) allows the microprocessor's status to be decoded to generate a refresh cycle for transparent refresh. If, for whatever reason, no external REFRQ is generated for 15 microseconds, the internally generated refresh will take over, so memory integrity will be guaranteed.

Arbitration

Because RAMs cannot do a read or write cycle and a refresh cycle at the same time, some form of *arbitration* must be provided to determine when refresh cycles will be performed.

Arbitration may be done by the microprocessor or by the dynamic RAM controller. Microprocessor arbitration may be implemented as follows:

A counter, running from the microprocessor's clock, is used to time the period between refresh cycles. At terminal count, the arbitration logic asserts the bus request signal to prevent the microprocessor from performing any more memory cycles. When the microprocessor responds with a bus grant, the arbitration logic generates a refresh cycle (or cycles, if burst refresh is

used). After refresh is complete, the arbitration logic releases the bus. This method has several disadvantages: First, time is wasted in exchanging bus control, which would not be required if the RAM controller did arbitration. Second, while refresh is being performed, *all* bus activity is stopped; for instance, even if the microprocessor is executing out of ROM at the time, it must stop until refresh is over. Third, bursts of DMA transfers must be kept very short, as refresh cannot be performed while DMA is in progress.

Some microprocessors, such as the Zilog Z-80, generate refresh cycles themselves after instruction fetches. This removes the need for external arbitration logic, but still has several disadvantages: First, DMA bursts still must be kept short to allow the CPU to do refresh. Second, this method adds to the complexity of the microprocessor, without removing the need for the RAM controller which is still required to do address multiplexing and $\overline{RAS}$, $\overline{CAS}$ and $\overline{WE}$ timing. Microprocessor refresh can cause problems of RAM compatibility; for instance, the Z-80 only outputs a 7-bit refresh address, which means some 64K RAMs which use 256 row refresh cannot be used with the Z-80. Also, since the Z-80 refresh cycle is a fixed length (no wait states), faster speed selections of the Z-80 are not compatible with slower dynamic RAMs. Third, systems employing multiprocessing or DMA are harder to implement, because of the difficulty in insuring the microprocessor will be able to perform refresh.

It is preferable to have arbitration performed by the dynamic RAM controller itself. This method avoids all the problems described above, but introduces a complication. If the microprocessor issues a read or write command while the dynamic RAM is in the middle of a refresh cycle, the RAM controller must make the microprocessor wait until it is done with the refresh before it can complete the read or write cycle. This means that from when the microprocessor activates the read or write signal, the time until the cycle can be completed can vary over a range of roughly 200 to 700 ns. Because of this, an acknowledge signal from the dynamic RAM controller is required to tell the microprocessor the memory cycle it requested is complete. This signal goes to the microprocessor's READY logic.

Memory Organization

As each dynamic RAM operates on only one bit at a time, multiple RAMs must be operated in parallel to operate on a word at a time. RAMs operated in this way are called a *bank* of RAM. A bank consists of as many RAMs as there are bits in the memory word. When used in this way, all address and control lines are tied to all RAMs in the bank.

A single bank of RAM will provide 64K words of memory in the case of the 2164A, or 16K words in the case of the 2118. To provide more memory words, multiple banks of RAM are used. In this case, all address, CAS, and $\overline{WE}$ lines are tied to all RAMs, but each bank of RAM has *its own* $\overline{RAS}$. Each bank knows whether it is being addressed during a read or write operation by whether or not its $\overline{RAS}$ input was activated — if not, then all other inputs are ignored during that cycle.

Data outputs for RAMs in corresponding bit positions in each of the banks may be tied in common, since they are 3-state outputs; even though $\overline{CAS}$ is connected to all banks of RAM, only that bank whose $\overline{RAS}$ is active will enable its data outputs in response to $\overline{CAS}$ going active. Data inputs for RAMs in corresponding bit positions in each of the banks are also tied in common.

INTEL DYNAMIC RAM CONTROLLERS

The Intel 8202A and 8203 Dynamic RAM Controllers each provide all the interface logic needed to use dynamic RAMs in microprocessor systems, in a single chip. Either the 8202A or 8203 allow a dynamic RAM memory to be implemented using a minium of components, board space, and power, and in less design time than any other approach.

The following sections will describe each of these controllers in detail.

8202A

FUNCTIONAL DESCRIPTION

The 8202A provides total dynamic RAM control for 4K and 16K dynamic RAMs, including the Intel 2104A, 2117, and 2118. The pinout and simplified logic diagram of the 8202A are shown in Figures 10 and 11.

The 8202A is always in one of the following states:

a) IDLE
b) TEST cycle
c) REFRESH cycle
d) READ cycle
e) WRITE cycle

The 8202A is normally in the idle state. Whenever a cycle is requested, the 8202A will leave the idle state to perform the desired cycle; if no cycle requests are pending, the 8202A will return to the idle state. A refresh cycle request may originate internally or externally to

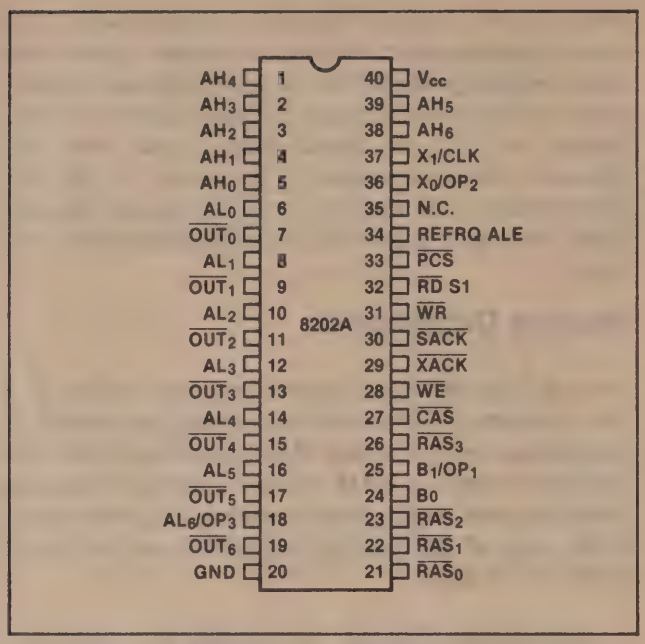


Figure 10. 8202A Pinout

the 8202A; all other requests come only from outside the 8202A.

A test cycle is requested by activating the $\overline{RD}$ and $\overline{WR}$ inputs simultaneously, independent of $\overline{PCS}$ (Protected Chip Select). The test cycle will reset the refresh address counter to zero and perform a write cycle. A test cycle should not be allowed to occur in normal system operation, as it interferes with normal RAM refresh.

A refresh cycle performs a $\overline{RAS}$-only refresh cycle of the next lower consecutive row address after the one previously refreshed. A refresh cycle may be requested by activating the REFRQ input to the 8202A; this input is latched on the next 8202A clock. If no refresh cycles are requested for a period of about 13 microseconds, the 8202A will generate one internally. By refreshing one row every 15.6 microseconds or sooner, all 128 rows will be refreshed every 2 ms. Because refresh requests are generated by the 8202A itself, memory integrity is insured, even if the rest of the system should halt operation for an extended period of time.

The arbiter logic will allow the refresh cycle to take place only if there is not another cycle in progress at the time.

A read cycle may be requested by activating the $\overline{RD}$ input, with $\overline{PCS}$ (Protected Chip Select) active. In the Advanced Read mode, a read cycle is requested if the microprocessor's S1 status line is high at the falling edge of ALE (Address Latch Enable) and $\overline{PCS}$ is active. If a dynamic RAM cycle is terminated prematurely, data loss may result. The 8202A chip select is "protected" in that once a memory cycle is started, it will go to completion, even if the 8202A becomes de-selected.

A write cycle may be requested by activating the $\overline{WR}$ input, with $\overline{PCS}$ active; this is the same for the normal and Advanced Read modes.

BLOCK DIAGRAM

Let's look at the detailed block diagram in Figure 12 to see how the 8202A satisfies the interface requirements of the dynamic RAM.

Address Multiplexing

Address multiplexing is achieved by a 3-to-1 multiplexer

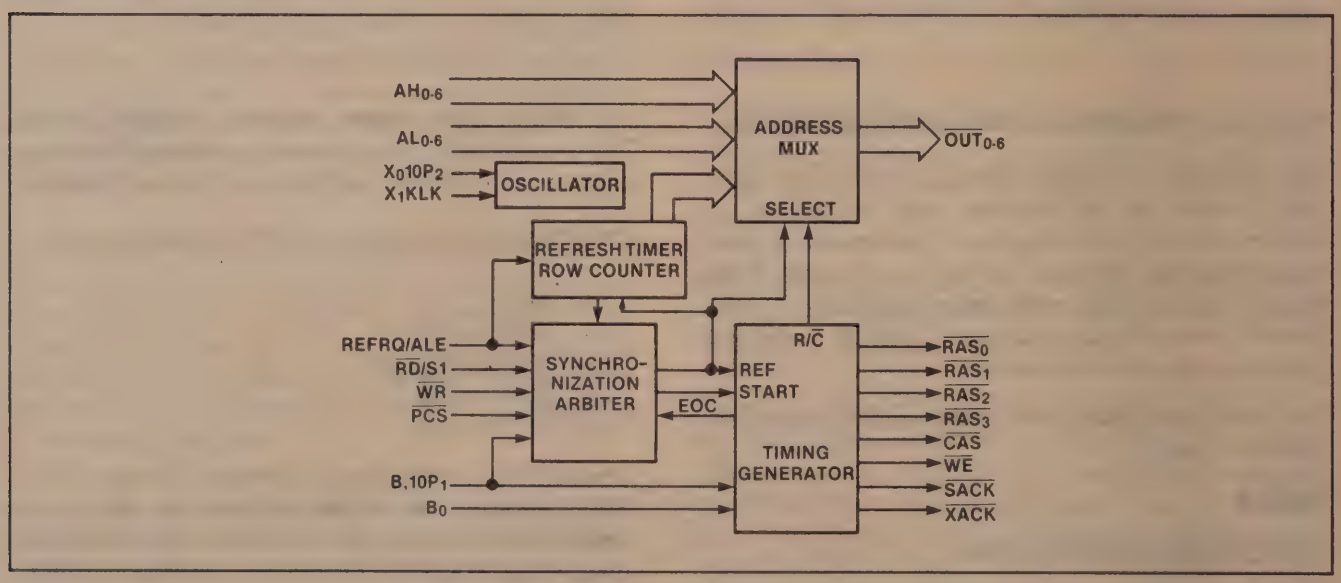


Figure 11. 8202A Simplified Block Diagram

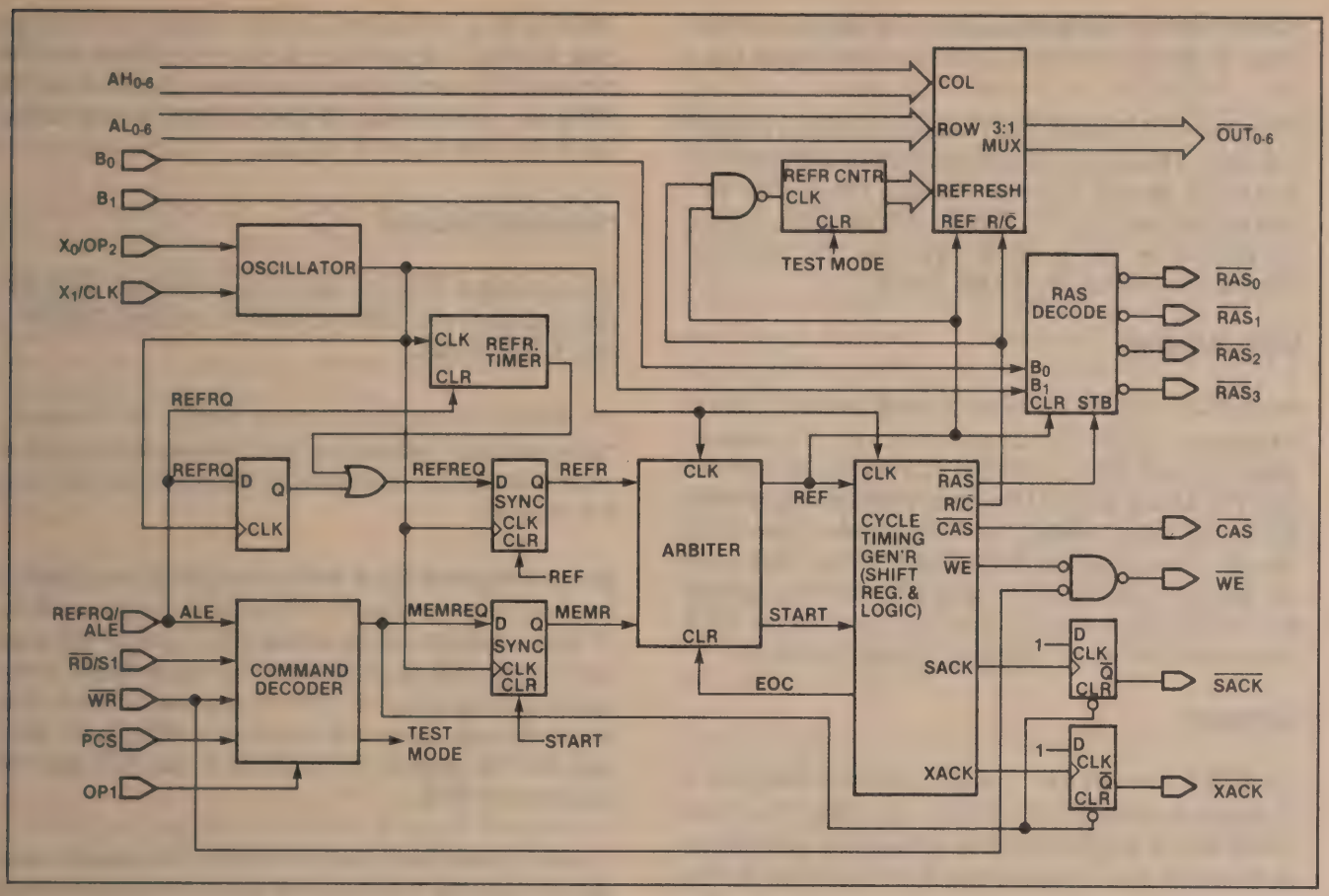


Figure 12. 8202A Detailed Block Diagram

internal to the 8202A; the three inputs are the row address (AL_{0-6}), column address (AH_{0-6}), and refresh row address (generated internally). When the 8202A is in the Idle state, the multiplexer selects the row address, so it is prepared to start a memory cycle. If a refresh cycle is requested either internally or externally, the address multiplexer will select the refresh row address long enough before $\overline{RAS}$ goes active to satisfy the RAM's t_{ASR} parameter.

To minimize propagation delays, the 8202A address outputs ($\overline{OUT}_{0-6}$) are inverted from the address inputs. This has no effect on RAM operation; inverters are not needed on the address outputs.

Doing this multiplexing internally minimizes timing skews between the address, $\overline{RAS}$, and $\overline{CAS}$, and allows higher performance than would otherwise be possible.

Refresh Counter

The next row to be refreshed is determined by the refresh counter, which is implemented as a 7-bit ripple-carry counter. During each refresh cycle, the counter is

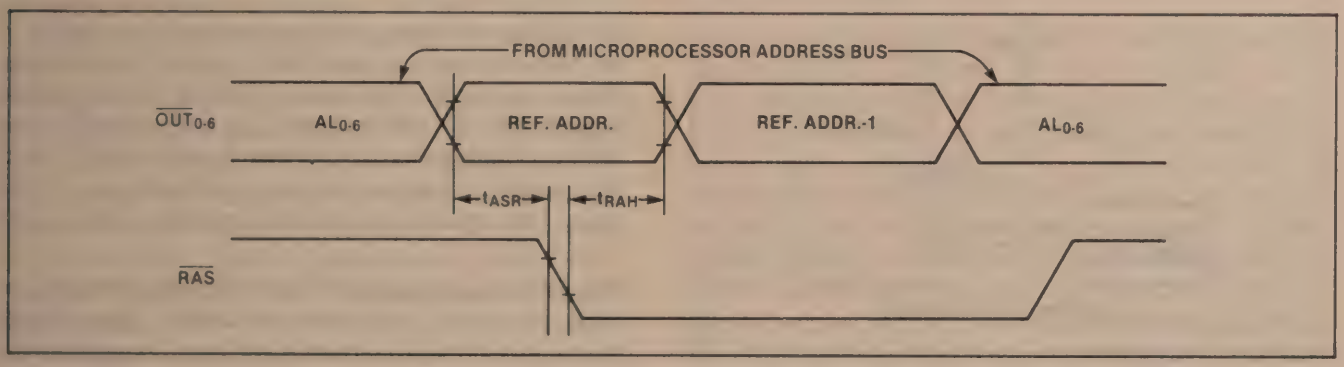


Figure 13. Detailed 8202A Refresh Cycle

incremented by one in preparation for the next refresh cycle (a refresh cycle is shown in detail in Figure 13).

When the 8202A enters TEST mode, the refresh counter is cleared. This feature is useful for automatic testing of the refresh counter function. Because the address outputs are inverted, the first refresh address after clearing the counter in test mode is $7F_H$, and the addresses *decrease* for subsequent refresh cycles.

$\overline{RAS}$ Decoding

Which bank of RAM is selected for a memory cycle is determined by the $\overline{RAS}$ decoder from the $B_{0\text{-}1}$ inputs, which normally come from the microprocessor address bus. The 8202A Timing Generator produces an internal $\overline{RAS}$ pulse which strobes the $\overline{RAS}$ decoder, generating the appropriate external $\overline{RAS}$ pulse. The $B_{0\text{-}1}$ inputs are *not* latched, so they must be held valid for the length of the memory cycle. During a refresh cycle, all the $\overline{RAS}$ outputs are activated, refreshing all banks at once.

Oscillator

The 8202A operates from a single reference clock with a frequency between 18.432 MHz and 25 MHz; this clock is used by the synchronization, arbitration, and timing generation logic. This clock may be generated by an on-board crystal oscillator, or by an external TTL-compatible clock source. When using the internal oscillator (available only on part number D8202A-1 or D8202A-3), a fundamental-mode crystal is attached to pins 36 and 37 (X_0 and X_1), as shown in Figure 14. The external TTL clock option is selected by pulling pin 36 (OP_2) to +12v through 1K ohm resistor, and attaching the clock input to pin 37 (CLK).

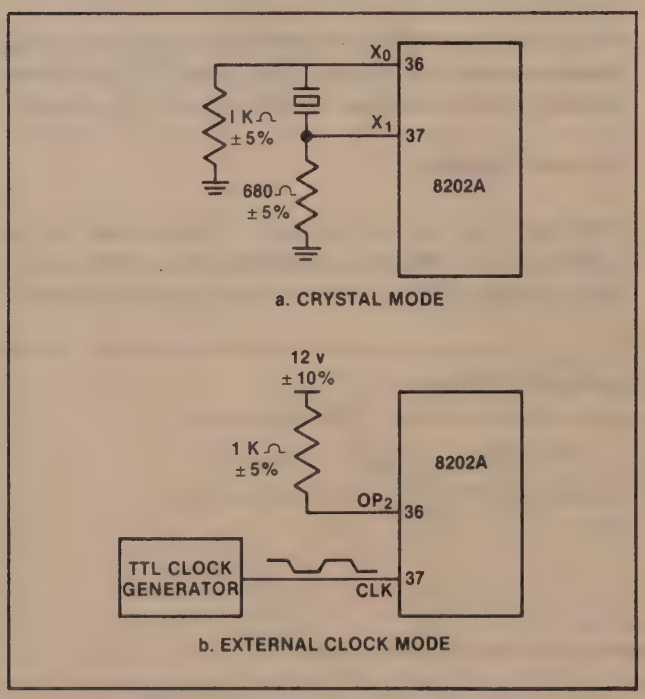


Figure 14. 8202A Clock Options

Command Decoder

The command decoder takes the commands from the bus and generates internal memory request (MEMR), and TEST signals.

The 8202A has two bus interface modes: the "normal" mode, and the "Advanced Read" mode. In the normal mode, the 8202A interfaces to the usual bus RD and WR signals.

In the Advanced Read mode, the 8202A interfaces to the Intel microprocessor bus signals ALE, S1, and $\overline{WR}$. S1 must be high on the falling edge of ALE for read cycles, and $\overline{WR}$ must be low for write cycles (write cycles are the same as for normal read mode). The 8085A S1 may be used directly by the 8202A; the 8086 and 8088 $\overline{S1}$ must be inverted. ALE and $\overline{WR}$ must be qualified by $\overline{PCS}$.

The Advanced Read mode is useful for reducing read data access time, and thus wait states. This mode is used mainly with 8085A systems.

If both $\overline{RD}$ and $\overline{WR}$ are active at once (regardless of the state of $\overline{PCS}$), the internal TEST signal is generated and the 8202A performs a test cycle as described above. One or both of $\overline{RD}$ and $\overline{WR}$ should have pull-up resistors to prevent the 8202A from inadvertantly being put into test mode, as the $\overline{RD}$ and $\overline{WR}$ signals are 3-stated by the microprocessor when RESET or HOLD are active. Since the test mode resets the refresh address counter, the refresh sequence will be interrupted, and data loss may result.

Refresh Timer and REFRQ

The 8202A contains a counter, operated from the internal clock to time the period from the last refresh cycle. When the counter times out, an internal refresh request is generated. This refresh period is proportional to the 8202A's clock period, and varies from 10.56 to 15.625 microseconds. Even at the lowest refresh rate, all the rows of the dynamic RAM will be refreshed every 2 ms.

The 8202A has an option of reducing the refresh rate by a factor of two, for use with 4K RAMS. These RAMs have only 64 rows to refresh every 2 ms, so need refresh cycles only half as often. This option is selected by pull-

ing pin 18 (AL_6/OP_3) to +12v through a 5.1K ohm resistor. This pin normally serves as the high-order row address input for the address multiplexer, but it is no longer needed for this function, as 4K RAMs have one less address input.

A refresh cycle may also be requested externally by activating the REFRQ input. This input is latched, so it only needs to be held active a maximum of 20 ns. If the 8202A is currently executing a memory cycle, it will complete that cycle, and then perform the refresh cycle. The internal and external refresh requests are ORed together before going to the arbiter.

The REFRQ input cannot be used in the Advanced Read mode, as the REFRQ pin is used for ALE in this mode.

REFRQ is most often used to implement transparent refresh, as explained in the section *Dynamic RAMS — Refresh*. This technique is not useful in iAPX 86 and iAPX 88 systems, so REFRQ is normally tied to ground.

The refresh timer is reset as soon as a refresh cycle is started (whether it was requested internally or externally). The time between refresh cycle (t_{REF}) is measured from when the first cycle is *started*, not when it was *requested*, which occurs sometime earlier. Of course, t_{REF}min does not apply if REFRQ is used — you may externally request refresh cycles as often as you wish.

Arbiter

This is the hardest section of a dynamic RAM controller to implement. If a read or write arrives at the same time as a refresh request, the arbiter must decide which one to service first. Also, if a read, write, or refresh request arrives when another cycle is already in progress, the arbiter must delay starting the new cycle until the current cycle is complete.

Both of the internal signals REFR (refresh request) and MEMR (memory cycle request) are synchronized by D-type master-slave flip-flops before reaching the arbiter. these circuits have been optimized to resolve a valid logic state in as short a time as possible. Of course, with any synchronizer, there is a probability that it will fail — not be able to settle in one logic state or the other in the allowed amount of time, resulting in a memory failure — but the 8202A has been designed to have less than one system memory failure every three years, based on operation in the worst case system timing environments.

Both synchronizers and the arbiter are operated from the 8202A's internal clock. Assuming the 8202A is initially in an idle state, one full clock period after the synchronizers sample the state of the MEMREQ and REFREQ signals, the arbiter examines the REFR and MEMR outputs of the synchronizers. If MEMR is active, the arbiter will activate START to begin the memory cycle (either read or write) on that clock. If REFR is active (regardless of the state of MEMR), the arbiter will activate START and REF to begin a refresh cycle on that clock. Once the cycle is complete, the Cycle Timing Generator will generate an end-of-cycle (EOC) signal to clear the arbiter and allow it to respond to any new or pending requests on the next clock.

Once a memory cycle is started, it cannot be stopped, regardless of the state of the $\overline{RD}$/S1, $\overline{WR}$, $\overline{ALE}$, or $\overline{PCS}$ inputs. This is necessary, as ending a dynamic RAM cycle prematurely may cause loss of data. Note, however, that the RAM $\overline{WE}$ output is directly gated by the $\overline{WR}$ input, so if $\overline{WR}$ is removed prematurely, the RAM $\overline{WE}$ pulse-width spec (t_{WP}) may be violated, causing a memory failure.

What happens if a memory request and refresh request occur simultaneously?

> If the 8202A is in the idle state, the *memory* request will be honored first.
>
> If the 8202A is *not* in the idle state (a memory or refresh cycle is in progress) then the memory cycle will lose priority and the refresh cycle will be honored first.

Remember, if the 8202A is performing a cycle, the arbiter doesn't arbitrate again until the end of that cycle. So the memory and refresh cycles are "simultaneous" if they both happen early enough to reach the arbiter before it finishes the current cycle. This arbitration arrangement gives memory cycles priority over refresh cycles, but insures that a refresh cycle will be delayed at most one RAM cycle.

Refresh Lock-Out

As a result of the 8202A operation, transparent refresh circuits like the one shown in Figure 15 should not be used. This circuit uses the RD input, with some qualifying logic, to activate REFRQ whenever the microprocessor does an opcode fetch. This circuit will work fine, as long as the 8202A never has to generate an internal refresh request, which is unlikely (if nothing else, the system RESET pulse is probably long enough that the 8202A will throw in a couple of refreshes while the microprocessor is reset). If the 8202A ever does generate its own refresh, there is a probability that the microprocessor will try to fetch an opcode while the

refresh is still in progress. If that happens, the 8202A will finish the refresh, see both the RD and REFRQ inputs active, honor the REFRQ first, and start a *second* refresh. In the meantime, the microprocessor is sitting in wait states, waiting for the 8202A to complete the opcode fetch. When the 8202A finishes the second refresh, it will see both RD and REFRQ active again, and will start a *third* refresh, etc. The system "locks up" with the microprocessor sitting in wait states *ad infinitum*, and the 8202A doing one refresh cycle after another.

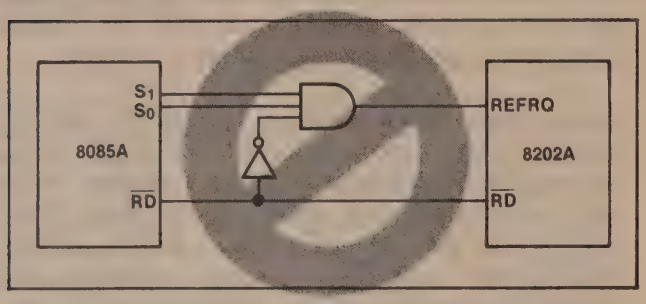


Figure 15. Improper Transparent Refresh Generation

To prevent this from happening, the transparent refresh circuit should be modified as shown in Figure 16. In this circuit, REFRQ cannot be activated until the opcode fetch is already in progress, as indicated by $\overline{SACK}$ being active (remember, $\overline{SACK}$ is never active during a refresh). If the microprocessor tries to do an opcode fetch while the 8202A is doing a refresh, REFRQ will not be active; the 8202A will finish the refresh and see only $\overline{RD}$ active, and will start the opcode fetch; only *then* will REFRQ be activated.

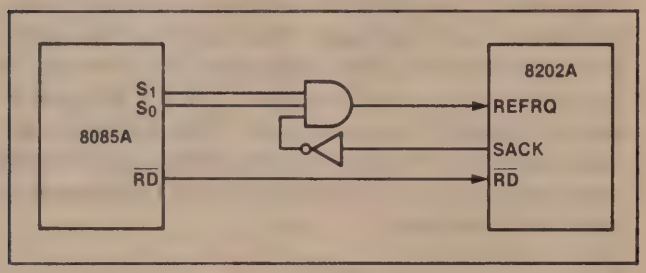


Figure 16. Generating Transparent Refresh For 8085A Systems

Cycle Timing Generator

The Cycle Timing Generator consists of a travelling-ones shift register and combinational logic required to generate all the RAM control signals and $\overline{SACK}$ and $\overline{XACK}$. All timings are generated from the 8202A's internal clock; no external delay lines are ever needed. The timing of these signals relative to CLK is illustrated in Figure 17.

When the cycle is complete, the Cycle Timing Generator sends an end-of-cycle (EOC) pulse to the arbiter to enable it to respond to new or pending cycle requests.

Minimum and maximum values for the 8202A parameters t_{CR} (Command to $\overline{RAS}$ active delay) and t_{CC} (Command to $\overline{CAS}$ active delay) differ by one 8202A clock period. This is because the commands ($\overline{RD}$, $\overline{WR}$, ALE) must be synchronized to the 8202A's clock; this introduces a ± one clock period (t_p) uncertainty due to the fact that the command may or may not be sampled on the first clock after it goes active, depending on the set-up time. If $\overline{RD}$ or $\overline{ALE}$ and $\overline{WR}$ are synchronous to the 8202A's clock, and the set-up time (t_{SC}) is met, the smaller number of clock periods will apply.

All 8202A output timings are specified for the capacitive loading in the data sheet. Typical output characteristics are shown in the data sheet for capacitive loads ranging from 0 to 660 pF, these can be used to calculate the effect of different loads than those specified in the data sheet on output timings. All address, $\overline{RAS}$, $\overline{CAS}$, and $\overline{WE}$ drivers are identical, so these characteristic curves apply to all outputs.

$\overline{SACK}$ AND $\overline{XACK}$

Because refresh cycles are performed asynchronously to the microprocessor's operation (except during transparent refresh), the microprocessor cannot know when it activates $\overline{RD}$ or $\overline{WR}$ if a refresh cycle is in progress, and therefore, it can't know how long it will take to complete the memory cycle.

This added consideration requires an acknowledge or "handshake" signal from the 8202A to tell the microprocessor when it may complete the memory cycle. This acknowledge would be used to generate the microprocessor's READY input — the microprocessor will sit in wait states until the 8202A acknowledges the memory cycle. Two signals are generated for this purpose by the 8202A; they are called *system acknowledge* ($\overline{SACK}$) and *transfer acknowledge* ($\overline{XACK}$). They serve the same purpose but differ in timing.

$\overline{XACK}$ is a Multibus-compatible signal, and is not activated until the read or write cycle has been completed by the RAMs. In a microprocessor system, however, there is a considerable delay from when the 8202A acknowledges the memory cycle until the microprocessor actually terminates the cycle. This delay is due to the time required to combine this acknowledge with other sources of READY in the system, synchronize READY to the microprocessor's clock, sample the state of READY, and respond to an active READY signal. As a result, more wait states than necessary may actual-

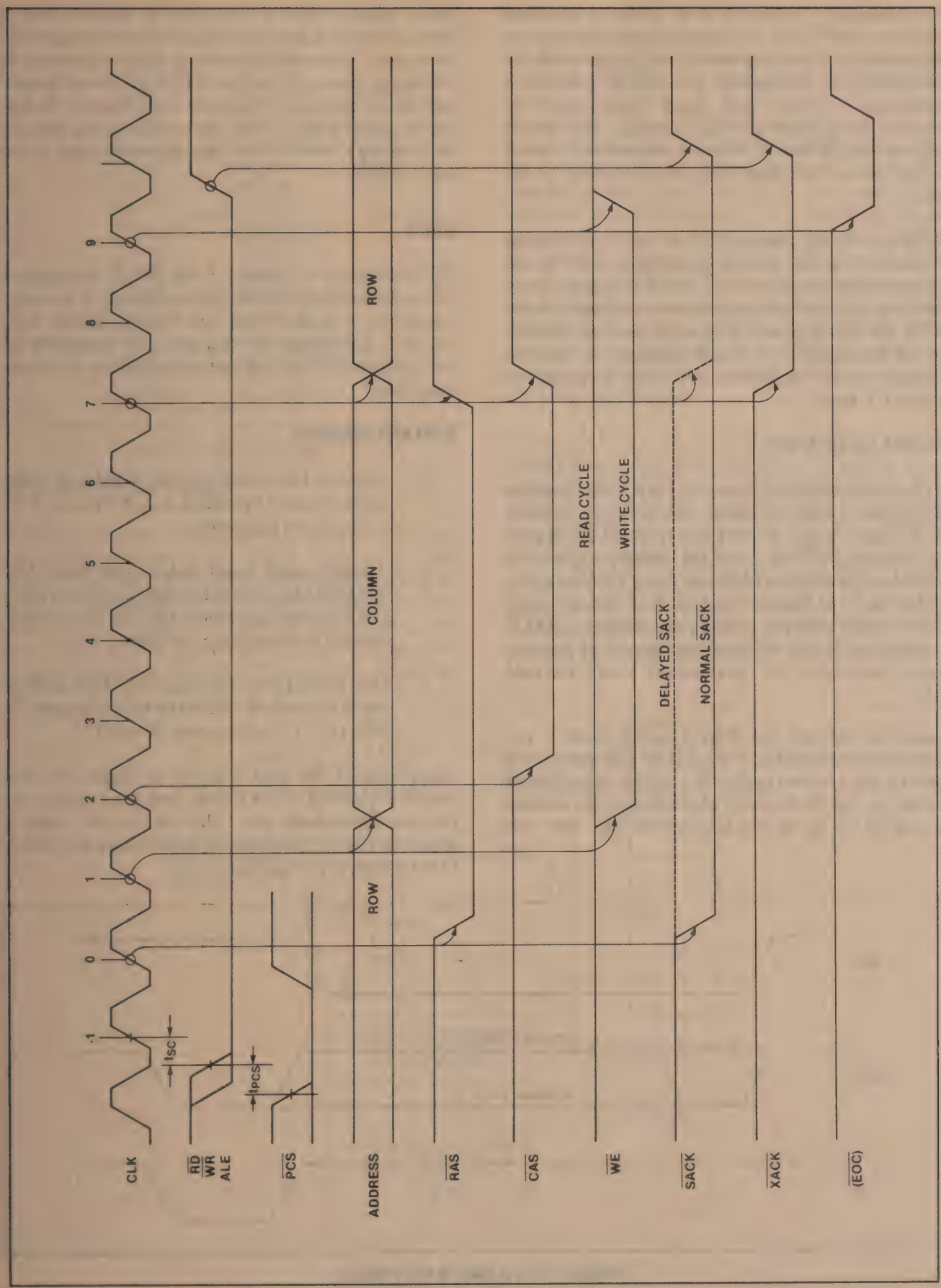


Figure 17. 8202A Timing Relative To CLK

ly be generated by using $\overline{XACK}$. $\overline{SACK}$ is activated earlier in the cycle to improve performance of microprocessors by compensating for the delays in the microprocessor responding to $\overline{XACK}$, and thus eliminating unneeded wait states which might be generated as a result of $\overline{XACK}$ timing. The system designer may use one or the other acknowledge signal, or use both in different parts of the system, at his option.

$\overline{SACK}$ and $\overline{XACK}$ are activated by the Cycle Timing Generator, but they can be de-activated only by the microprocessor removing its $\overline{RD}$ or $\overline{WR}$ request, or by activating ALE when in the advanced read mode. As the $\overline{SACK}$ and $\overline{XACK}$ signals are used to generate READY for the microprocessor, this is necessary to give the microprocessor as much time as it needs to respond to its READY input.

Delayed SACK Mode

$\overline{SACK}$ may be activated at one of two different times in the memory cycle; the earlier case is called "normal $\overline{SACK}$" and the later is called "delayed $\overline{SACK}$" (Figure 18). Delayed $\overline{SACK}$ occurs if the memory request was received by the 8202A while it was doing a refresh cycle. In this case, the memory cycle will be delayed some length of time while the refresh cycle completes; $\overline{SACK}$ is delayed to ensure the microprocessor will generate enough wait states. This is a concern mostly for read cycles.

Because of the way the delayed $\overline{SACK}$ mode is implemented in the 8202A, if the $\overline{RD}$ or $\overline{WR}$ input is activated while a refresh cycle is in progress, regardless of whether or not the 8202A is chip-selected, the internal delayed $\overline{SACK}$ mode flip-flop will be set. The next 8202A memory cycle will have $\overline{SACK}$ delayed, even if that cycle was not actually delayed due to a refresh cycle in progress. The delayed SACK flip-flop will be reset at the end of that cycle, and the 8202A will return to normal $\overline{SACK}$ operation. The same thing happens in Advanced Read mode if S1 is high at the falling edge of ALE during a refresh cycle, once again regardless of the state of $\overline{PCS}$.

8203

The 8203 is an extension of the 8202A architecture which allows the use of 64K dynamic RAMs. It is pinout compatible with the 8202A and shares identical A.C. and D.C. parameters with that part. The description of the 8202A applies to this part also, with the modifications below.

ENHANCEMENTS

1. Supports 16K *or 64K* dynamic RAMs. 4K RAM mode, selected by pulling AL_6/OP_3 (pin 18) to +12v, is not supported.

2. Allows a single board design to use either 16K or 64K RAMs, without changing the controller, and only making between two and four jumper changes to reconfigure the board.

3. May operate from external TTL clock without the +12v pull-up which the 8202A requires (a +5v or +12v pull-up may be used).

The pinout of the 8203 is shown in Figure 19. This pinout is identical to the 8202A, with the exception of the five highlighted pins. The function of these is described below. The simplified block diagram is similar to the 8202A's, in Figure 11.

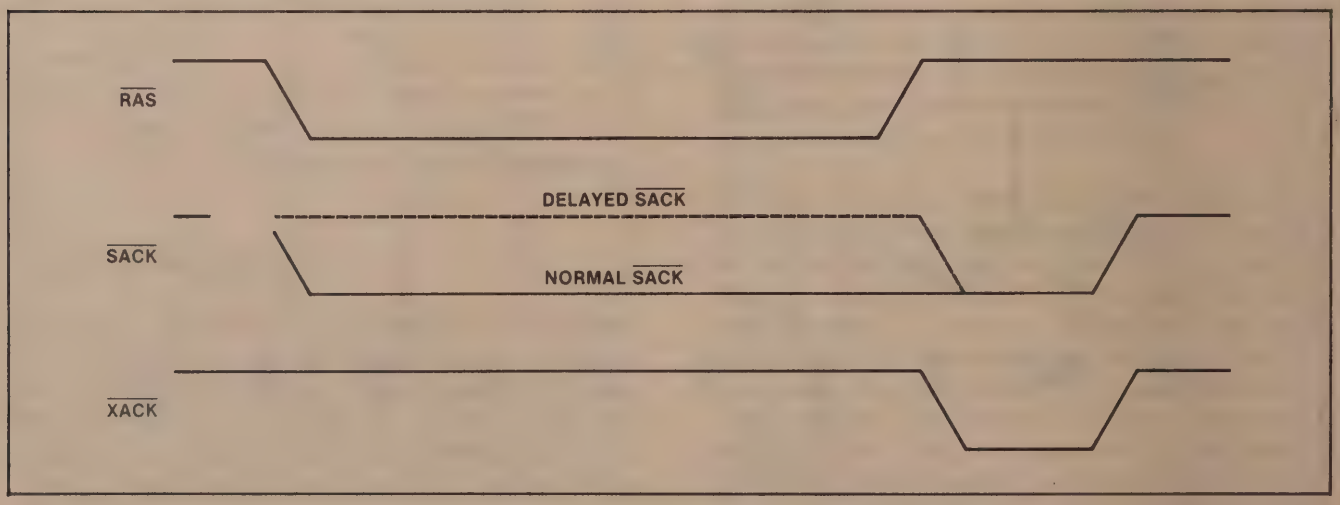


Figure 18. Delayed SACK Mode

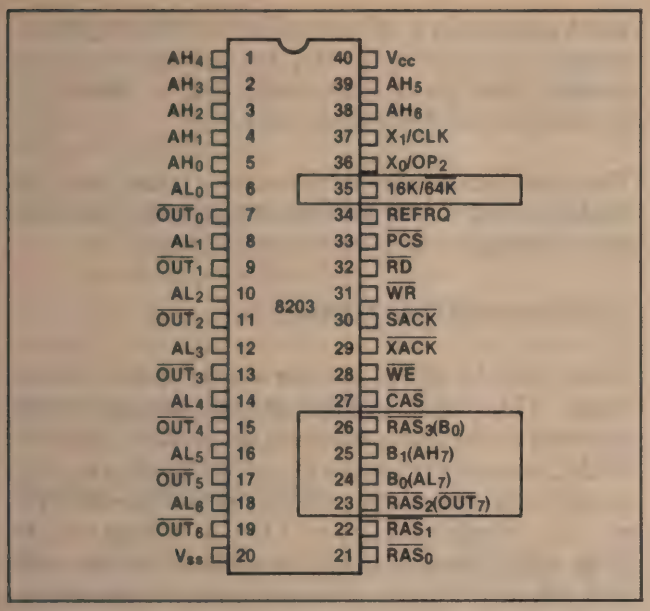


Fig. 19 8203 Pinout

16K Mode and 64K Mode

The goal of the 8203 is to provide a pin- and timing-compatible upgrade of the 8202A for use with 64K RAMs. The difficulty in doing this is that 64K RAMs require an additional address input compared to 16K RAMs, and thus the 8203 needs three more pins (one more RAM address output, and two more inputs to its internal address multiplexer). Since all but one of the 8202A's pins are already used, this is clearly a challenge — some functionality must be sacrificed to gain 64K RAM support. The 8203 reduces the maximum number of banks supported from four to two for 64K RAMs.

Pin 35 (16K/$\overline{64K}$) is used to tell the 8203 whether it is being used to control 16K RAMs or 64K RAMs. When tied to V_{cc} or left unconnected, the 8203 operates in the 16K RAM mode; in this mode all the remaining pins function identically to the 8202A. When tied to ground, it operates in the 64K RAM mode, and pins 23 through 26 change function to enable the 8203 to support 64K RAMs. Pin 35 (16K/$\overline{64K}$) contains an internal pull-up —when unconnected, this input is high, and the 8203 operates identically to the 8202A. This maintains pinout compatibility with the 8202A, in which pin 35 is a no-connect, so the 8203 may be used in 8202A sockets with no board modifications.

When the 8203 is in the 64K RAM mode, four pins change function, as shown in Table 2. The pins change function in this particular way to allow laying out a board to use either 16K or 64K RAMs with a minimum of jumpers, as shown in Figure 20. This figure shows the 8203 with two banks of RAM. Banks 0 and 1 may be either 16K RAMs or 64K RAMs; banks 2 and 3 may only be 16K RAMs, as the 8203 supports two banks of 64K RAM. For clarity, only those connections which are important in illustrating the 8203 jumper options are shown.

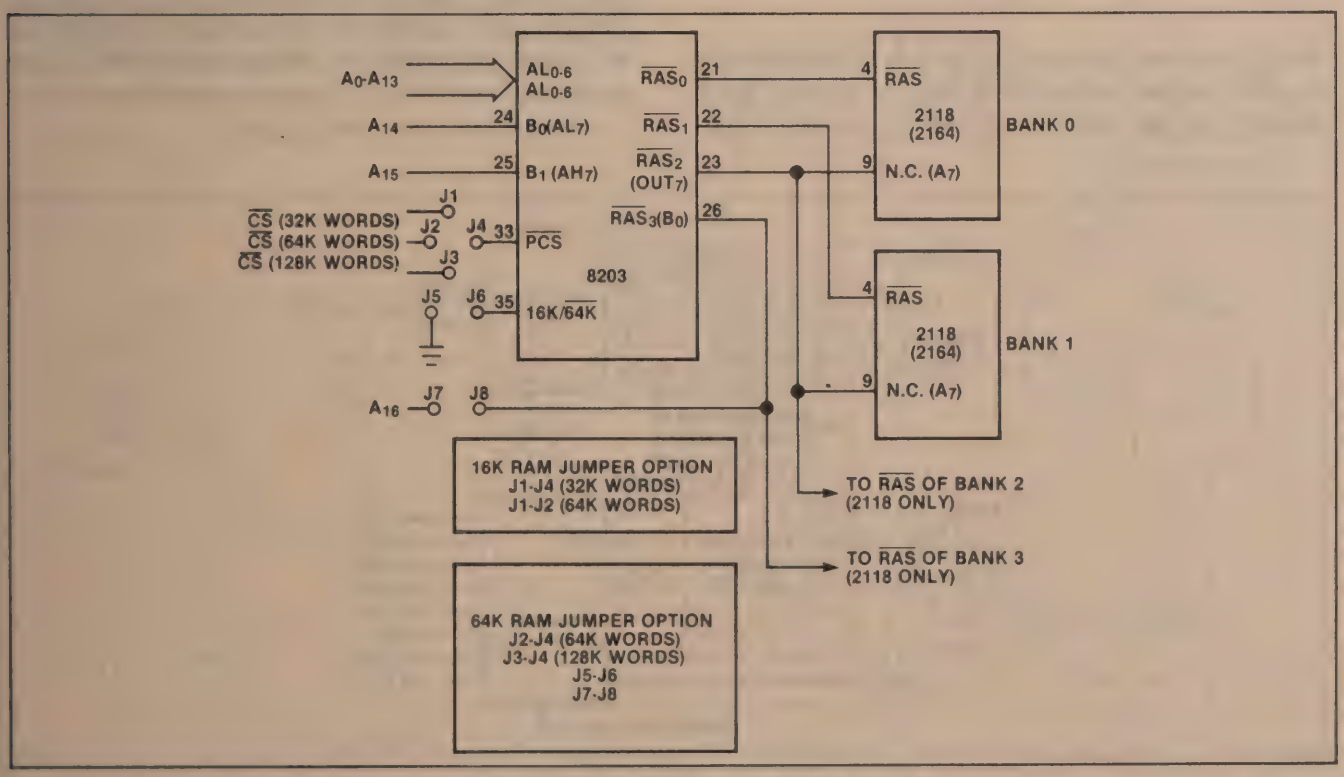


Figure 20. 8203 Jumper Options

Table 2. 16K/$\overline{64K}$ Mode Selection

Pin #	16K Function	64K Function
23	$\overline{RAS}_2$	Address Output (OUT_7)
24	Bank Select (B_0)	Address Input (AL_7)
25	Bank Select (B_1)	Address Input (AH_7)
26	$\overline{RAS}_3$	Bank Select (B_0)

Jumpers J1-J4 may be used to chip select the 8203 over various address ranges. For example, if two banks of 16K RAMs are replaced with two banks of 64K RAMs, the address space controlled by the 8203 increases from 32K words to 128K words. If four banks of 16K RAMs are replaced with one bank of 64K RAMs, no chip select jumpers are needed.

In the 64K RAM mode, pins 24 and 25 ($B_0(AL_7)$ and $B_1(AH_7)$) change function from bank select inputs to address inputs for the 64K RAM. Since the bank select inputs normally come from the address bus anyway, no jumper changes are required here. The bank select function moves to pin 26 ($\overline{RAS}_{3(B_0)}$); since only two bank of 64K RAM is supported, only one bank select input is needed in this mode, not two. Jumpers J6 and J7 are shorted in the 64K RAM mode to connect pin 26 (B_0) to the address bus. In the 16K RAM mode, these jumpers must be disconnected, as pin 26 junctions as the $\overline{RAS}_3$ output; in the 64K RAM mode, this bank is not populated, so $\overline{RAS}_3$ is not needed.

Pin 23 serves two functions: in the 16K RAM mode it is the $\overline{RAS}$ output for bank 2 ($\overline{RAS}_2$), in the 64K RAM mode is the high order RAM address output ($\overline{OUT}_7$), which goes to pin 9 of the 64K RAMs. This requires no jumpers as when using 16K RAMs, pin 9 is a no-connect, and when using 64K RAMs, bank 2 is depopulated, so $\overline{RAS}_2$ is not used.

This arrangement allows converting a board from 16K RAMs to 64K RAMs with no change to the controller and changing a maximum of three jumpers.

+5v External Clock Option

Just as with the 8202A, the user has the option of an external TTL clock instead of the internal crystal oscillator as the timing reference for the 8203; unlike the 8202A, he does not need to tie pin 36 ($X_0/0P_2$) to +12v to select this option—this pin may be tied to *either* +5v *or* +12v. If pin 36 is tied to +12v, a 1K ohm (± 5%) series resistor must be used, just as for the 8202A. If pin 36 is tied to +5v, it must be tied *directly* to pin 40 (V_{cc}) with no series resistor. This is because pin 36 must be within one Schottky diode voltage drop (roughly 0.5v) of pin 40 to select the external TTL clock option; a series resistor may cause too great a voltage drop for the external clock option to be selected. For the same reason, the trace from pin 36 to 40 should be kept as short as practical.

Test Cycle

An 8203 test cycle is requested by activating the $\overline{RD}$, $\overline{WR}$, and $\overline{PCS}$ inputs simultaneously. By comparison, an 8202A test cycle requires activating only the $\overline{RD}$ and $\overline{WR}$ inputs simultaneously, independent of $\overline{PCS}$. Like the 8202A, and 8203 test cycle resets the address counter to zero and performs a write cycle.

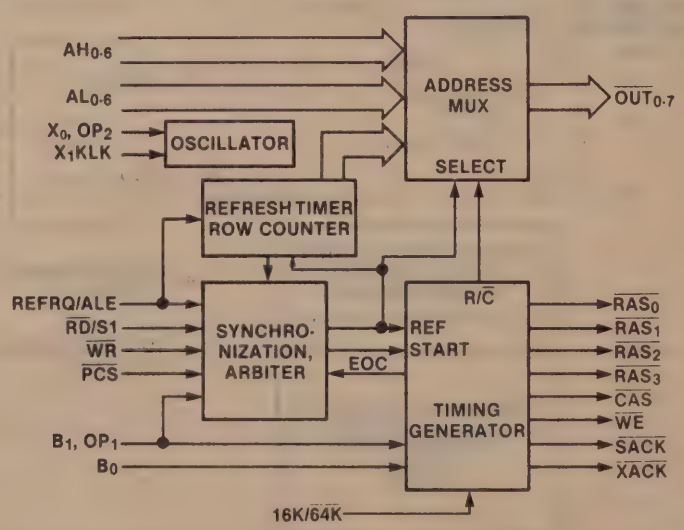


Figure 21. 8203 Simplified Block Diagram

BLOCK DIAGRAM

A simplified block diagram of the 8203 is shown in Figure 21. It is identical to the 8202A except for the following differences:

1. The 3:1 address multiplexer is 8 bits wide, instead of 7 bits wide, to support the addressing requirements of the 64K RAM.
2. The refresh address counter is 8 bits. This allows it to support RAMs which use either the 128-row or 256-row refresh schemes. Regardless of which type of RAM is used, the refresh counter cycles through 256 rows every 4 ms. RAMs which use 128-row re-fresh treat the eighth address bit as a "don't care" during refresh, so they see the equivalent of 128-row refresh every 2 ms. In either case the rate of internally-generated refresh cycles is the same—at least one every 15.6 microseconds.

INTEL iAPX-86 AND iAPX-88

Device Descriptions

The iAPX-86 and iAPX-88 are advanced 16-bit microprocessor families, based on the 8086 and 8088 microprocessors, respectively. While both have a similar architecture and are software compatible, the 8086 transfers data over a 16-bit bus, while the 8088 uses an 8-bit data bus (but has a 16-bit internal bus).

Min and Max Modes

In order to support the widest possible range of applications, the 8086 and 8088 can operate in one of two modes, called minimum and maximum modes. This allows the user to define certain processor pins to "tailor" the 8086 or 8088 to the intended system. These modes are selected by strapping the MN/MX (minimum/maximum) input pin to V_{cc} or ground.

In the minimum mode, the microprocessor supports small, single-processor systems using a minimum of components. In this mode, the 8086 or 8088 itself generates all the required bus control signals (Figure 22).

In the maximum mode, the microprocessor supports larger, higher performance, or multiprocessing systems. In this mode, the 8086 or 8088 generates status outputs which are decoded by the Intel 8288 Bus Controller to provide an extensive set of bus control signals, and Multibus compatibility (Figure 23). This allows higher performance RAM operation because the memory read and write commands are generated more quickly than is possible in the minimum mode. The maximum mode is the one most often used in iAPX-86 and iAPX-88 systems.

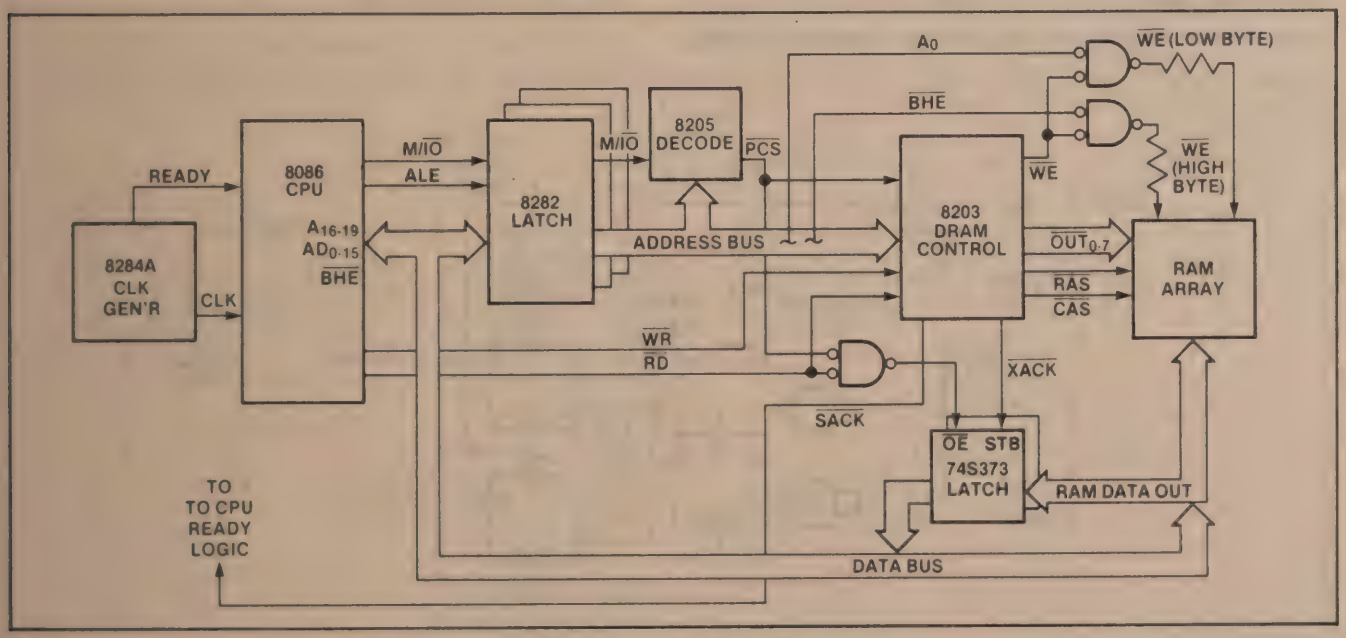


Figure 22. 8086 Minimum Mode

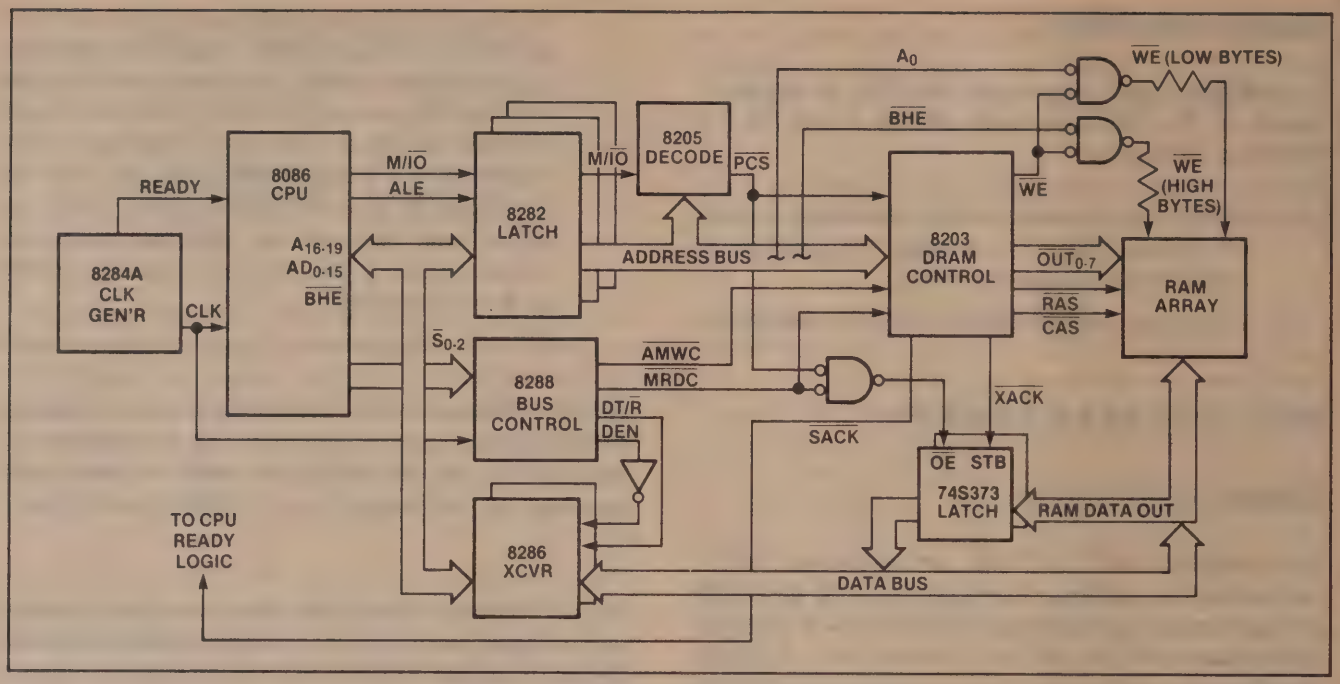


Figure 23. 8086 Maximum Mode

Alternate Configuration

The Alternate Configuration is not an operating mode of the 8086 or 8088 *per se*, but uses TTL logic along with the status outputs of the microprocesor to generate the RAM read and/or write control signals (Figure 24). The alternate configuration may be used with the microprocessor in either minimum or maximum mode. This configuration is advantageous because it activates the memory read and write signals even earlier than the maximum mode, leading to higher performance. It is possible to generate either the RAM read or write signal using this configuration, and generate the other RAM control signal using the min or max mode in the normal configuration.

Each of the three system configurations may be used with buffers on the address, data, or control bus for increased electrical drive capability.

Performance vs. Wait States

Before starting a discussion of timing analyses, it's worthwhile to look at the effect of wait states on the iAPX-86 and iAPX-88.

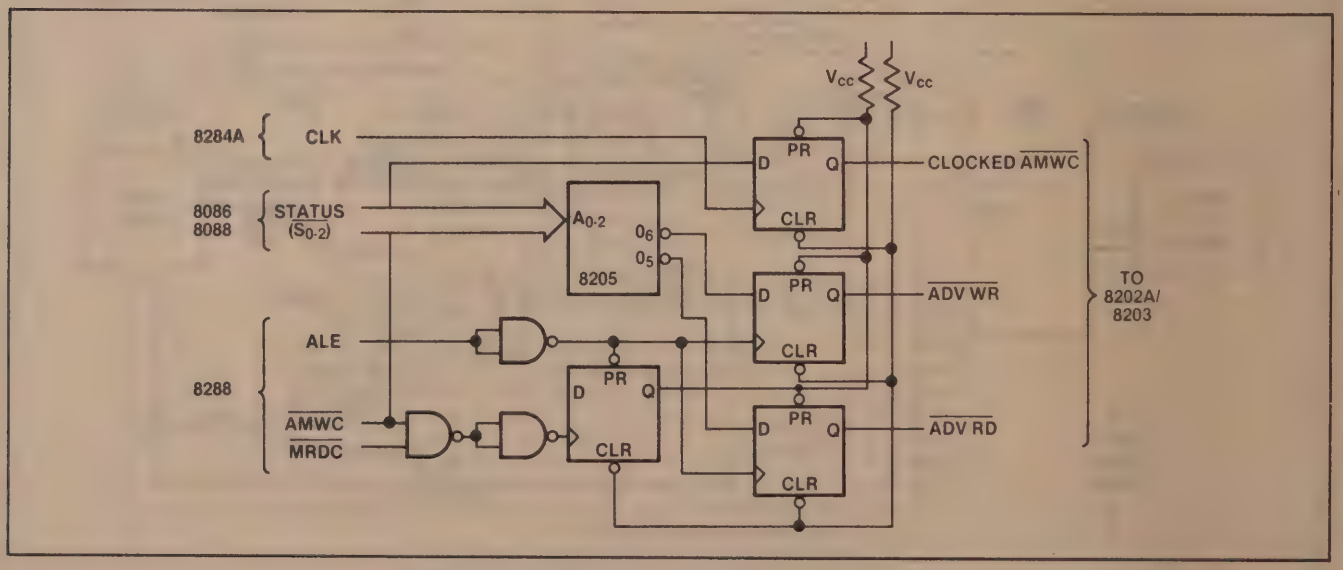


Figure 24. Alternate Configuration Logic

For most microprocessors, the effect of, say, one wait state on execution times is straightforward. If a bus cycle normally is three clocks long, adding a wait state to every bus cycle will make all bus cycles four clocks, decreasing performance by 33%. This is multiplied by the percentage of time that the microprocesor is doing bus cycles (some instructions take a long time to execute, so the microprocessor skips a few bus cycles).

The effect of wait states on the iAPX-86 and iAPX-88 is not so straightforward, however.

The 8086 and 8088 microprocessors consist of two processing units: the execution unit (EU) executes instructions, and the bus interface unit (BIU) fetches instructions, reads operands, and writes results. During periods when the EU is busy executing instructions, the BIU "looks ahead" and fetches more instructions from the next consecutive addresses in memory; these are stored in an internal queue. This queue is four bytes long for the 8088 and six bytes long for the 8086; under most conditions, the BIU can supply the next instructions without having to perform a memory cycle. Only when the program doesn't proceed serially (e. g. a Jump or Call instruction) does the EU have to wait for the next instruction to be fetched from memory. Otherwise, the instruction fetch time "disappears" as it is proceeding in parallel with execution of previously fetched instructions. The EU then has to wait for the BIU only when it needs to read operands from memory or write results to memorv. As a result, the 8086 and 8088 are less sensitive to wait states than other microprocessors which don't use an instruction queue. The effect of wait states on 8086 execution time compared to the Motorola 68000 and Zilog Z8000 for a typical mix of software is summarized in Table 3.[1]

Table 3. Effects of Wait States on Execution Time

	Execution Time Increase Over 0 Wait State Execution Time		
Processor	**1 Wait State**	**2 Wait States**	**3 Wait States**
iAPX 86/10 (measured)	8.3%	16.3%	26.3%
Z8000 (computed)	19.1%	38.2%	57.3%
68000 (computed)	15.9%	31.9%	47.8%

The BIU can fetch instructions faster than the EU can execute them, so wait states only affect performance to the extent that they make the EU wait for the transfer of operands and results. How much this affects program execution time is a function of the software; programs that contain many complex instructions like multiplies and divides and register operations are slowed down less than programs that contain primarily simple instructions. The effect of wait states on the 8086 and 8088 is always less than on other microprocessors which don't use an instruction queue.

[1] From *16-Bit Microprocessor Benchmark Report: iAPX-86, Z8000, and 68000,* publ. by Intel Corp. 1980

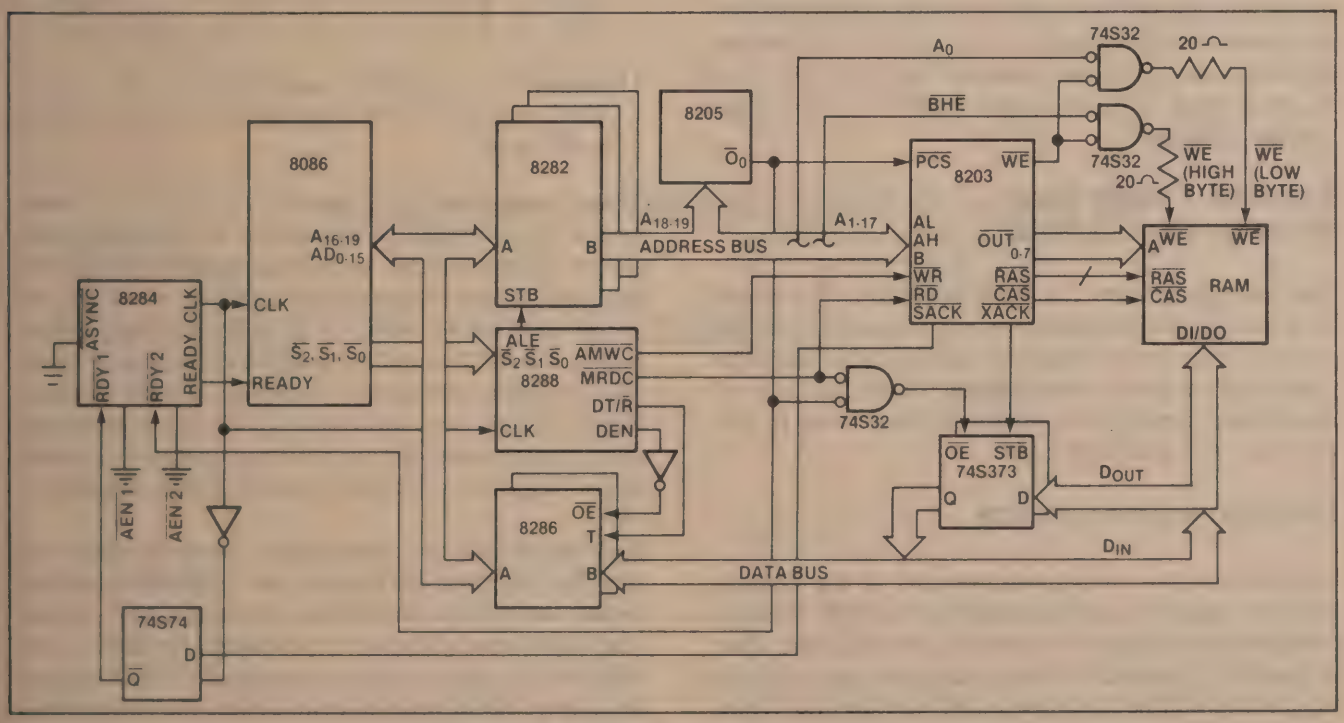


Figure 25. 8086 Max Mode System

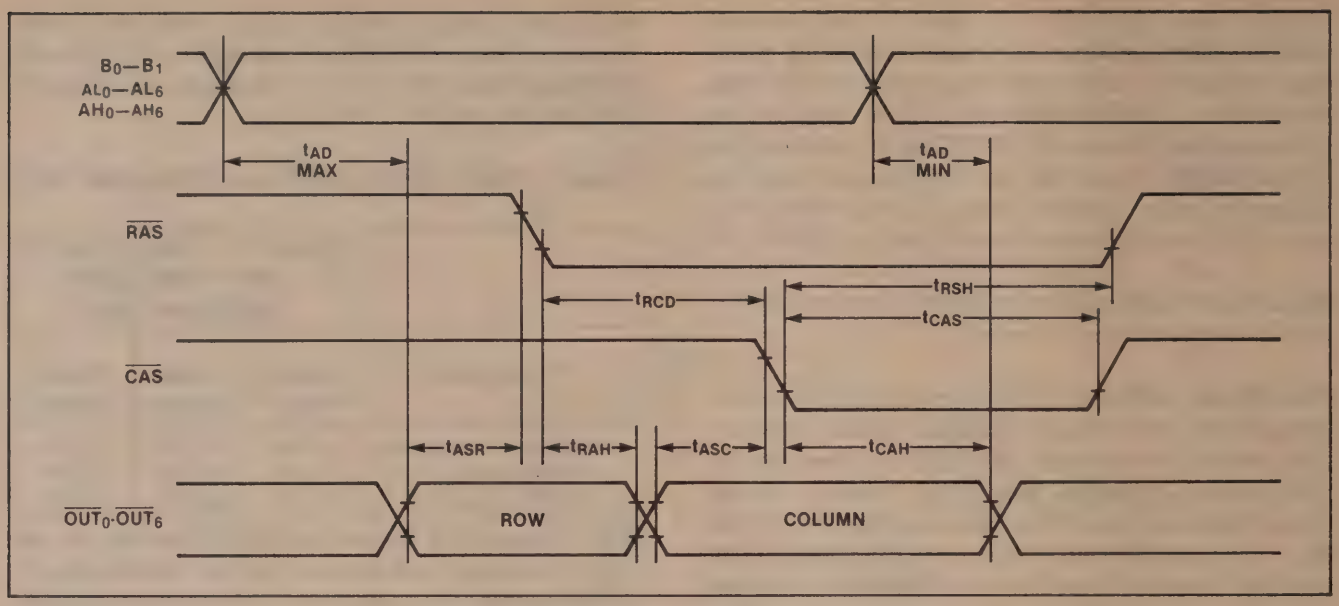


Figure 26. Memory Compatibility Timing

Timing Analysis

This section will look at two specific system configurations to show how the 8203 timing requirements are satisfied by the 8086. Methods of determining the worst case number of wait states for the various configurations are also given.

The timings of the 8202A and 8203 are identical; only the 8203 is referred to for the remainder of this note, but all comments apply equally to the 8202A. A11 timings are worst case over the range of $T_A = 0 - 70°C$ and $V_{cc} = +5v \pm 10\%$ for the test conditions given in the devices' data sheets.

Example 1. 8086 Max Mode System (5 MHz)

This example (Figure 25) is representative of a typical medium-size microprocesor system. Example 1 requires one wait state (worst case) for memory cycles. Example 2 also uses an 8086 in Max mode at 5 MHz, but uses external logic to reduce the number of wait states to zero for both read and write cycles.

DYNAMIC RAM INTERFACE

First, look at the timing requirements of the dynamic RAM to ensure they are satisfied by the 8203. Memory compatibility timings are shown in the 8203 data sheet (Figure 26). Seven 8203 timings are given, not counting t_{AD}, which will be discussed in the next section. These timings are summarized in Table 4.

Table 4. Memory Compatibility Timings
(all parameters are minimums)

Symbol	Parameter	Value
t_{ASC}	Column Address Set-Up Time	t_p-30
t_{ASR}	Row Address Set-Up Time	t_p-30
t_{CAH}	Column Address Hold Time	$5t_p$-30
t_{CAS}	$\overline{CAS}$ Pulse Width	$5t_p$-10
t_{RAH}	Row Address Hold Time	t_p-10
t_{RCD}[1]	$\overline{RAS}$ to $\overline{CAS}$ Delay Time	$2t_p$-40
t_{RSH}	$\overline{RAS}$ Hold Time from $\overline{CAS}$	$5t_p$-30

[1] $t_{RCD}min = t_{RAH}min + t_{ASC}min = 2_p - 40$
This parameter is the minimum RAS active to CAS active delay.

These timings are all a function of the 8203's clock period (t_p); they may be adjusted to be compatible with slower dynamic RAMs by slowing the 8203's clock (increasing t_p). The frequency of the 8203's clock may be varied from 18.432 MHz to 25 MHz; for best performance, the 8203 should be operated at the highest possible frequency compatible with the chosen dynamic RAM. In most cases, t_{RAH} or t_{CAS} will be the frequency limiting parameter, but the 8203 can operate at its maximum frequency with most dynamic RAMs available.

t_{ASR} applies only to refresh cycles. When the 8203 is in the Idle state (not performing any memory or refresh cycles) the address multiplexer allows the AL_{0-7} inputs (the RAM row address) to propagate through to the 8203 OUT_{0-7} pins, which are connected to the RAM address pins. So in read or write cycles, the row address will propagate directly from the address bus to the

RAM; the row address set-up time in this case is determined by the microprocessor's timing (see the next section). At the beginning of a refresh cycle, the 8203 has to switch its internal multiplexer to direct the refresh row address to the RAMs before activating RAS; the t_{ASR} parameter in Table 4 refers to this case only.

Assume the Intel 2164A-20 RAM (200 ns access time) is used. Equations 1(a)-(h) show that this RAM is compatible at the 8203's maximum operating frequency of 25 MHz (t_p = 1/(25 MHz) = 40 ns). This frequency will be used for now; once the rest of the system timings are calculated, the minimum 8203 frequency which will provide the same system performance can also be determined.

(a) $t_{ASC} = t_p - 30 = 10$ (Equation 1.)
(b) $t_{ASR} = t_p - 30 = 10$
(c) $t_{CAH} = 5t_p - 30 = 170$
(d) $t_{CAS} = 5t_p - 10 = 190$
(e) $t_{RAH} = t_p - 10 = 30$
(f) t_{RCD}[1] $= 2t_p - 40 = 40$
(g) $t_{RP} = 4t_p - 30 = 130$
(h) $t_{RSH} = 5t_p - 30 = 170$

[1] May be calculated as
$t_{RCD}min = t_{RAH}min + t_{ASC}min = 2t_p - 40$

ADDRESS SET-UP AND HOLD TIME MARGINS

The microprocessor must put the memory address on the address bus early enough in the memory cycle for it to pass through the 8203 and meet the row address set-up time to RAS (t_{ASR}) requirement of the dynamic RAM (Figure 27). Since the address propagates directly through the 8203, this set-up time is a function of how long the microprocessor holds the address on the bus before activating the RD or WR command, the address delay through the 8203 ($t_{AD}max$), and how long the 8203 waits before activating RAS ($t_{CR}min$). This is shown in Figure 28, and calculated in Equation 2. This and all following equations show timing margins; a positive result indicates extra margin, a zero result says the parameter is just met, and a negative result indicates it is not met for worst-case conditions.

Row Address Set-Up Time Margin (Equation 2.)

= CPU Address to $\overline{RD}$ Delay + $\overline{RAS}$ Active Delay - Address Delays

= TCLCL(5MHz) + TCLML min (8288) + t_{CR}min(8203) - [Greater of TCLAVmax(8086) + TIVOVmax (8282) or TCLLHmax(8288) + TSHOVmax(8282)] - t_{AD}max(8203) - t_{ASR}(2164A—20)

= 200 + 10 + [40 + 30] - [Greater of (110 + 30) or (15 + 45)] - 40 - 0

= 100

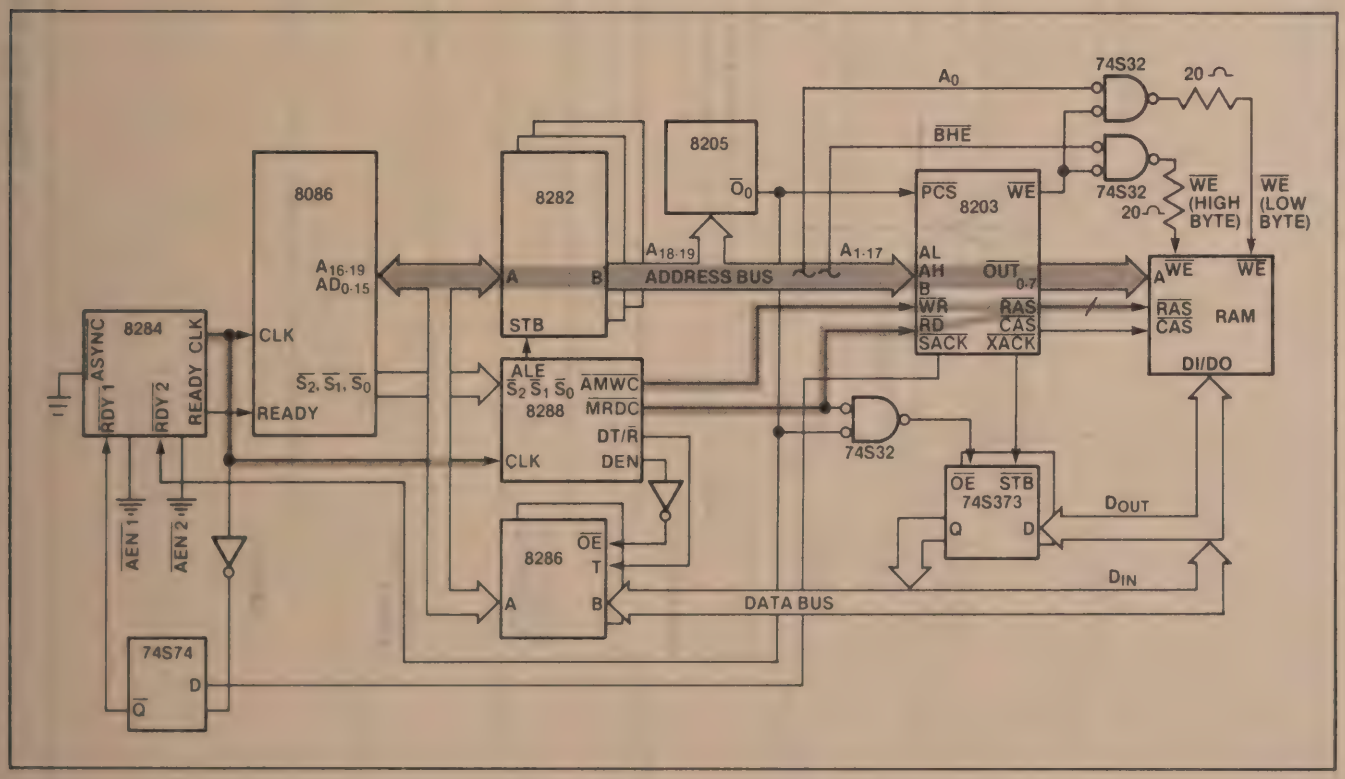


Figure 27. Address Set-Up and Hold Time Margins

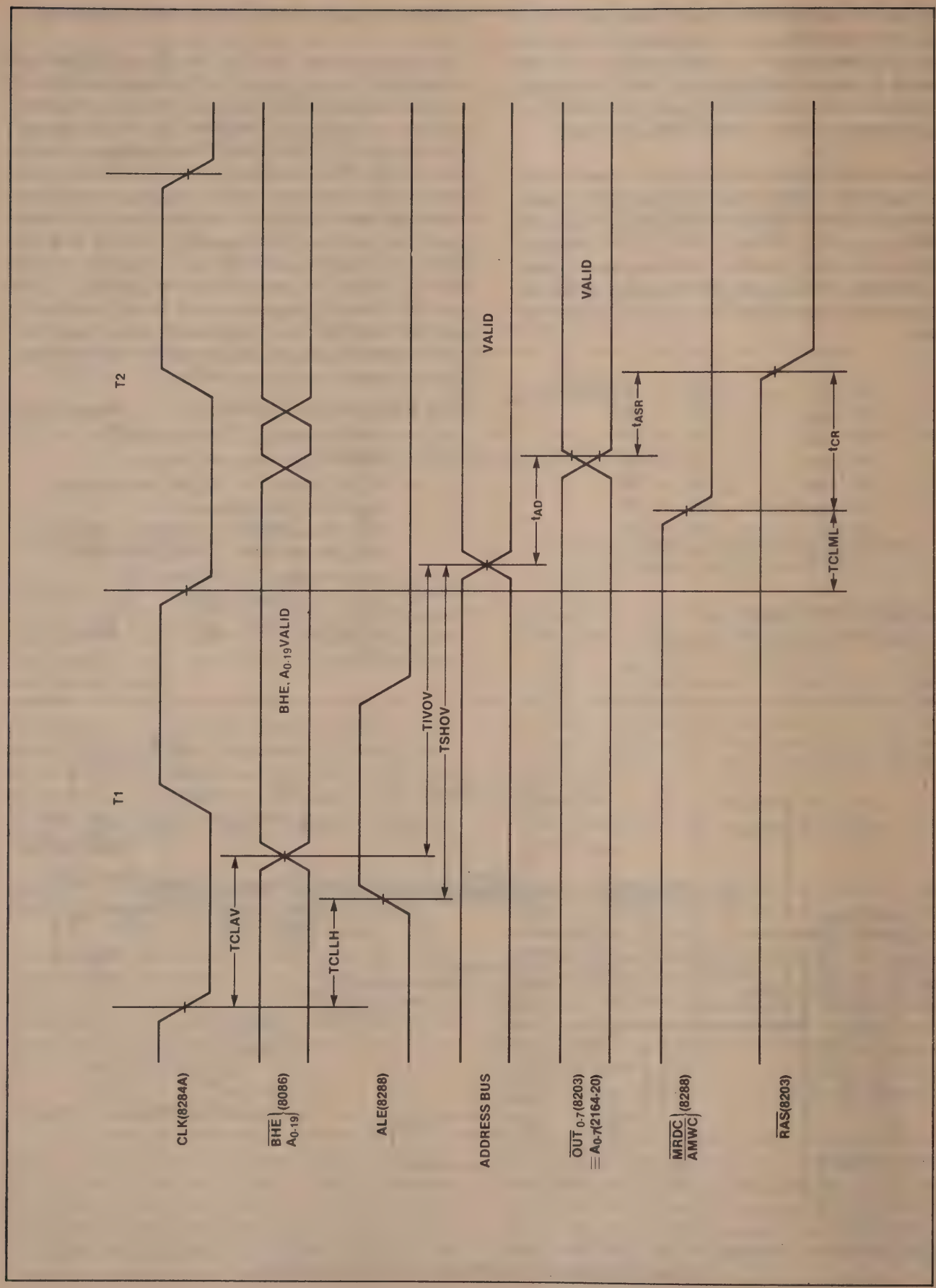


Figure 28. Address Set-up Time Margin

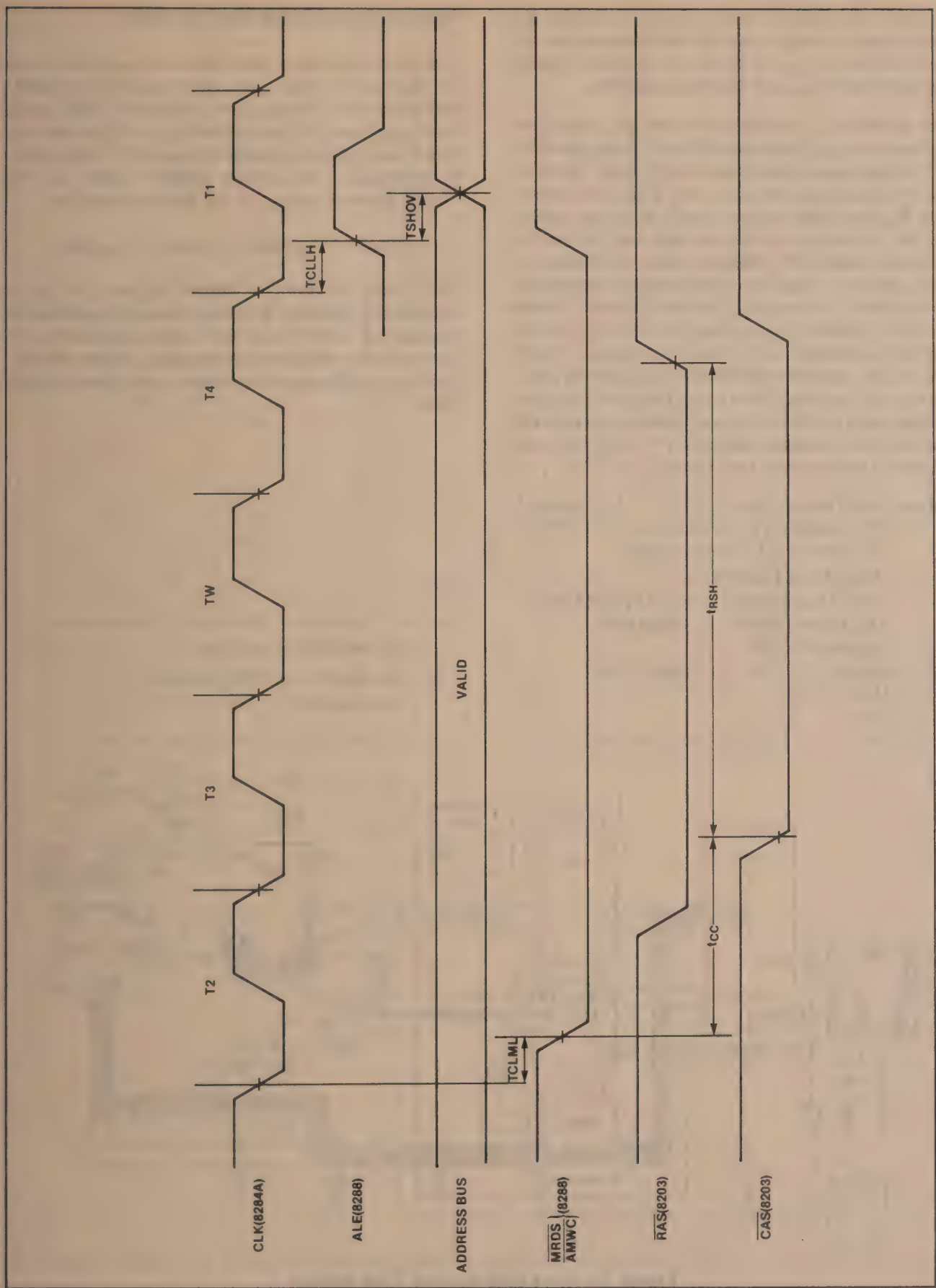


Figure 29. Address Hold Time Margin

Similarly, the microprocessor must maintain the memory address long enough to satisfy the column address hold time (t_{CAH}) of the RAM; the 8203 T_{AD}min parameter should be used for this calculation.

More importantly, the 8203 bank select (B_{0-1}) inputs are also not latched; these are used directly to decode which $\overline{RAS}$ output is activated during read or write cycles, so these inputs must be held valid until $\overline{RAS}$ goes inactive. Since B_{0-1} are usually taken directly from the address bus, this determines the address hold time required of the system (Figure 29). These are easily satisfied by the 8086 as shown by Equation 3. N represents the number of wait states. This equation can be tried with various values for N (starting with 0 and increasing) until the equation is satisfied, or it can be set equal to zero (meaning no excess margin remains) and solved for N directly; the fractional value for N that results must be rounded *up* to get the worst-case number of wait states to satisfy this particular parameter. No wait states are required to meet address hold times.

Address Hold Time Margin (N = 0) (Equation 3.)

= CPU Address Hold Time, from $\overline{RD}$ Active - $\overline{RAS}$ Inactive Delays

= (3+N)TCLCL(5MHz) + TCLLHmin(8288)[1] + TSHOVmin(8282)- TCLMLmax(8288) - t_{CC}mas(8203) - t_{RSH}max(8203)[2]

= 3(200) + 2 + 10 - 35 - [4(40) + 85] - [5(40) + 30]

= 102

READ DATA ACCESS TIME MARGIN

Read data access times determine how many wait states are required for read cycles. Remember that dynamic RAMs have two access time parameters, $\overline{RAS}$ access time (t_{RAC}) and $\overline{CAS}$ access time (t_{CAC}). Either one may be the limiting factor in determining RAM access time, as explained in the section *Dynamic RAM - Access Times*, above. Here t_{CAC} is the limiting factor, as

$$t_{CC}max + t_{CAC}max \geq t_{CR}max + t_{RAC}max.$$

This timing is shown in Figures 30 and 31, and is calculated in Equation 4. In this system, one wait state is required to satisfy the read data access time requirements of the system; the margin is -50 ns, which is too large a difference to be made up by using a faster RAM.

[1] Not specified — use 2 ns

[2] Not specified in 8203 data sheet; t_{RSH}max(8203) = $5t_p$ + 30

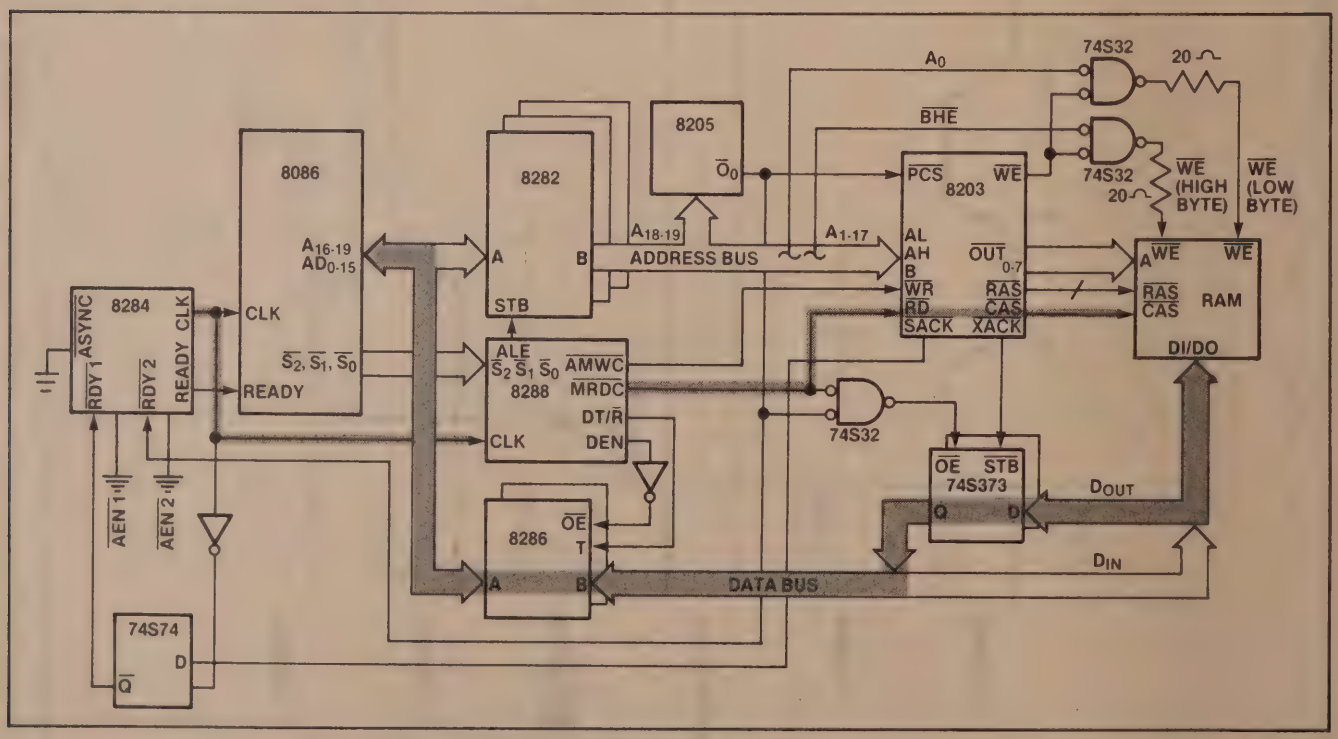


Figure 30. Read Data Access Time Margin

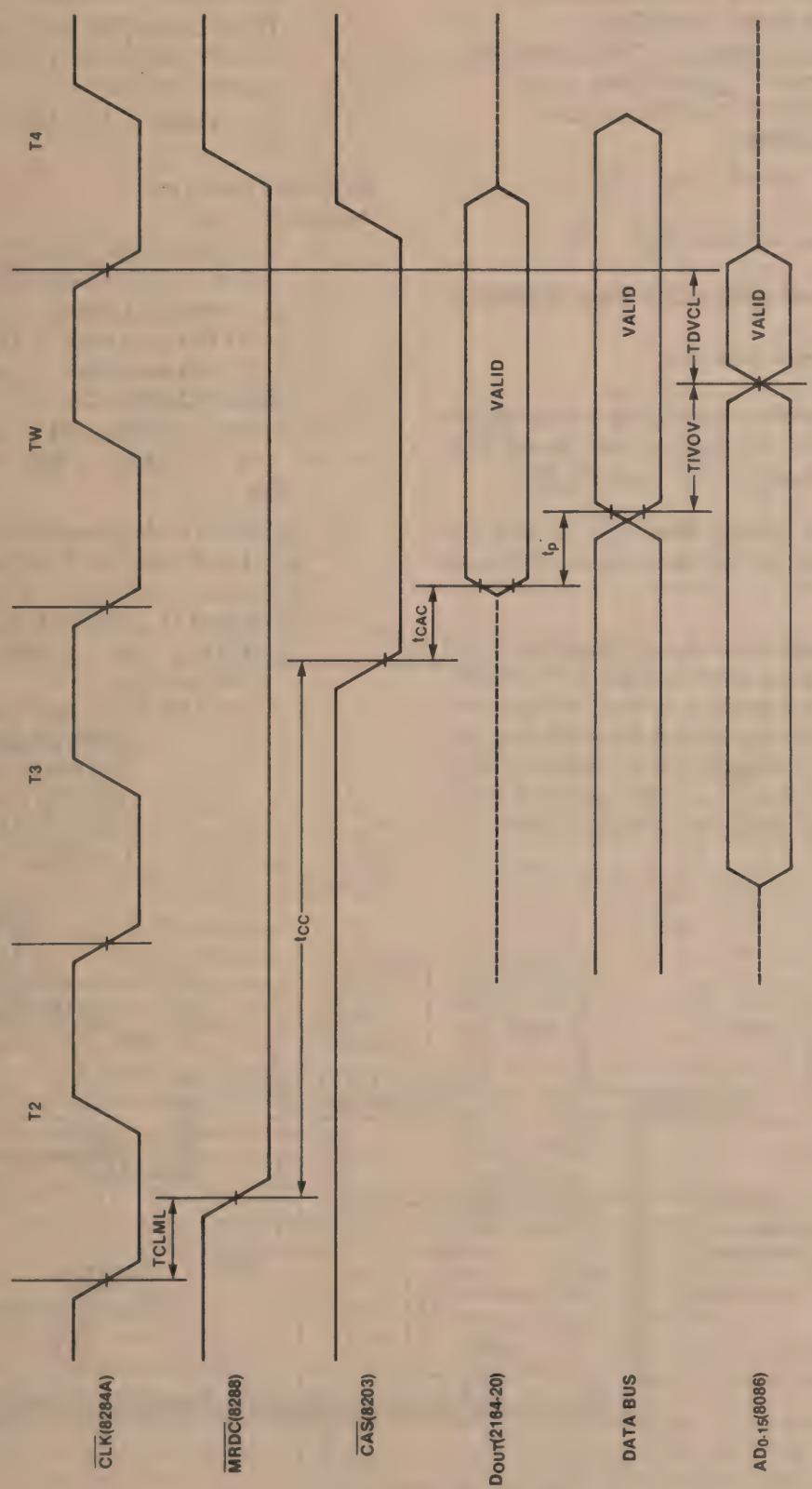


Figure 31. Read Data Access Time Margin

Read Data Access Time Margin (N = 0) (Equation 4.)

= CPU $\overline{RD}$ Active to Data Valid Delay - $\overline{CAS}$ Active Delay - Data Delays

= (2+N)TCLCL(5MHz) - TCLMLmax(8288) t_{CC}max(8203) - t_{CAC}max(2164A—20) - $t_P$max(74S373)[1] - TIVOVmax(8286) - TDVCLmin(8086)

= 2(200) - 35 - [4(40) + 85] - 110 - 30[1] - 30 - 30

= - 80 ⇒ 1 wait state needed (N = 1)

WRITE DATA SET-UP AND HOLD TIME MARGINS

In write cycles, the write data must

1. reach the dynamic RAMs long enough before $\overline{CAS}$ to meet the RAM's data set-up time parameter, t_{DS} (Figures 32 and 33), and

2. be held long enough after $\overline{CAS}$ to meet the RAM's data hold time parameter (t_{DH}) (Figures 32 and 34.)

Data set-up time margin is calculated in Equation 5, and data hold time margin is given in Equation 6. Again, these are margins, so a positive number indicates that system timing requirements are met for worst-case timings. Data hold time is a function of the number of 8086 wait states, represented as N, as is the read data access time margin. No wait states are required to meet this parameter.

Write Data Set-Up Time Margin (Equation 5.)

= CPU $\overline{WR}$ Active to Data Valid Delay + $\overline{CAS}$ Delay - Data Delay

= TCLMLmin(8288) + t_{CC}min(8203) - TCLDVmax(8086) - TIVOVmax(8286) - t_{DS}min(2164A-20)

= 10 + [3(40) + 25] - 110 - 30 - 0

= 15

Write Data Hold Time Margin (N = 0) (Equation 6.)

= CPU Data Hold Time, from $\overline{AMWC}$ Active + Data Delays - $\overline{CAS}$ Active Delay

= (2 + N)TCLCL(5MHz) + TCLCHmin(8284A) + TCHDXmin(8086) + TIVOVmin(8286) - TCLMLmax(8288) - t_{CC}max(8203) - t_{DH}min(2164A—20)

= 2(200) + [⅔(200) - 15] + 10 + 5 - 35 - [4(40) + 85] - 45

= 308

[1] t_p(74S373) is the greater of t_{PHL} (from data) or t_{PLH} (from data) and is compensated for V_{cc} and temperature variations, and is derated for a 300_pF load (T.I. spec is at 15_pF).

t_p(74S373) = 13ns + 0.05ns/$_p$F(300 - 15)$_p$F + 2.75ns = 30ns.

Where 13ns is T.I. spec value
0.05ns/$_p$F is derating factor for excess capacitive load
(300 - 15) is excess capacitive load 2.75 is compensation for T_A and V_{cc} variation

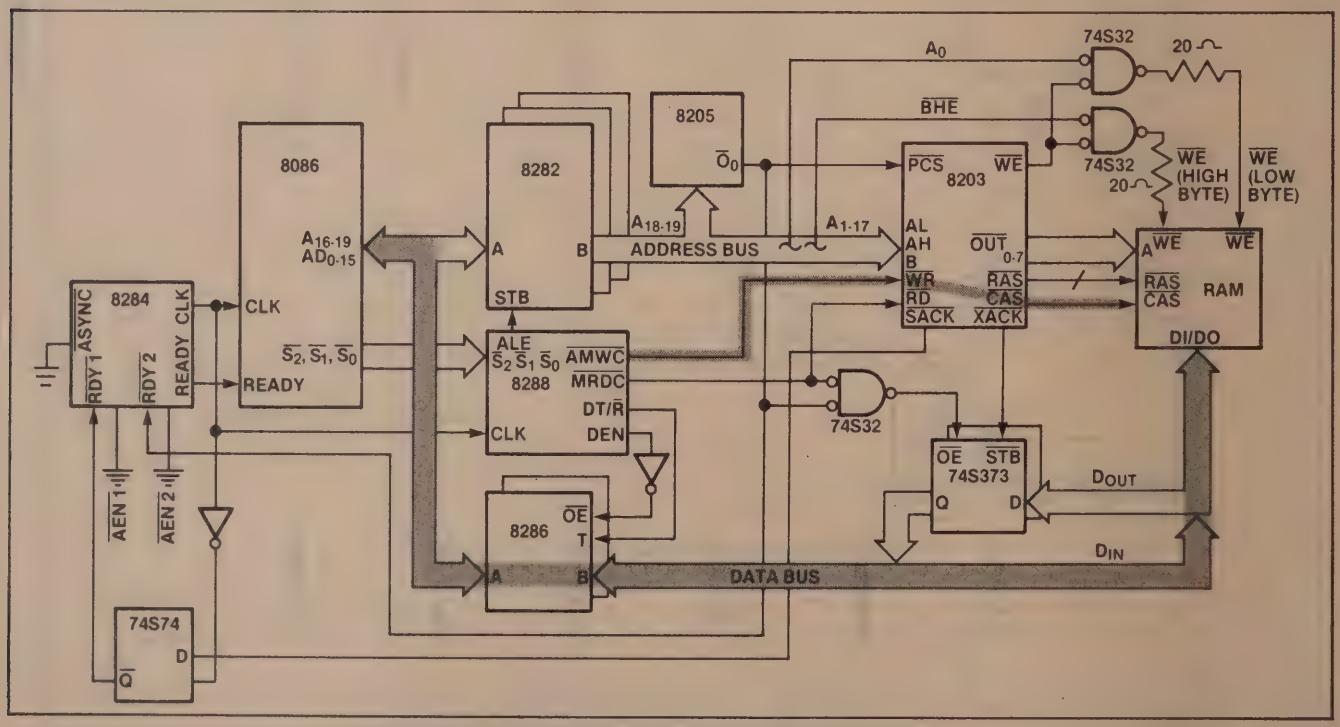


Figure 32. Write Data Set-Up and Hold Time Margins

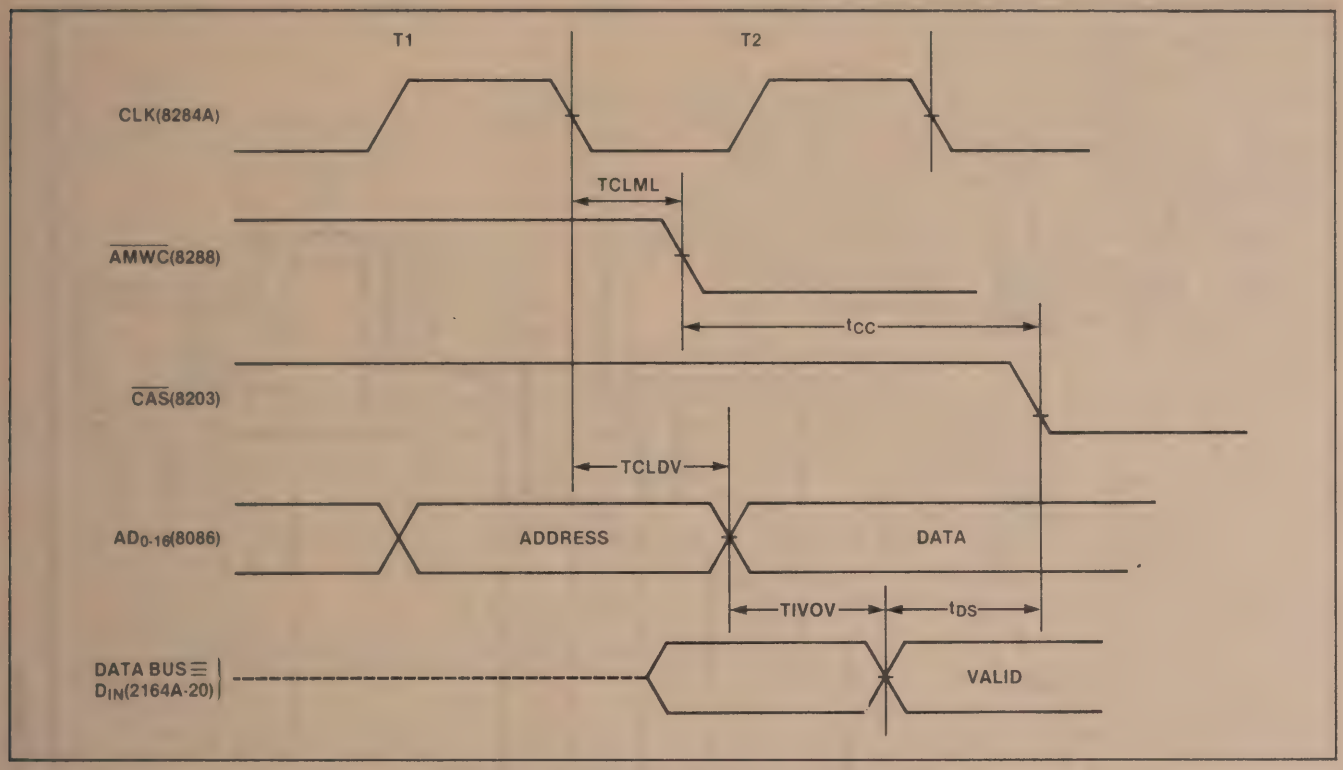


Figure 33. Write Data Set-Up Time Margin

$\overline{\text{SACK}}$ SET-UP TIME MARGIN

As explained earlier, $\overline{\text{SACK}}$ (and $\overline{\text{XACK}}$) are "handshaking" signals used to tell the microprocessor when it may terminate the bus cycle in progress. Thus, $\overline{\text{SACK}}$ timing determines how many wait states will be generated, as opposed to how many wait states are actually required for proper operation, which is determined by the read data access time for read cycles and by the write data hold time for write cycles. If $\overline{\text{SACK}}$ causes more wait states than are required, there is a performance penalty, but the system operates; if too few wait states are generated, the system will not function.

$\overline{\text{SACK}}$ and $\overline{\text{XACK}}$ serve the same function; they differ only in timing. $\overline{\text{XACK}}$ is Multibus compatible, and is activated only when the read data is actually on the bus (in a read cycle) or when the write data has been latched into the RAM (in a write cycle). $\overline{\text{SACK}}$ is activated earlier in the memory cycle than $\overline{\text{XACK}}$ to compensate for delays in the microprocessor responding to this signal to terminate the cycle. Use of $\overline{\text{SACK}}$ is normally preferable, as it results in the fewest possible wait states being generated. But in some systems, $\overline{\text{SACK}}$ will not generate a sufficient number of wait states, so $\overline{\text{XACK}}$ or a delayed form of SACK must be used. Note that the number of wait states generated by $\overline{\text{SACK}}$ and $\overline{\text{XACK}}$ will vary, depending on whether a refresh cycle is in progress when the memory cycle was requested, and if refresh cycle is in progress, how near it is to completion. $\overline{\text{SACK}}$ is sampled by the 8284A Clock Generator Chip's RDY1 or RDY2 input. The 8284A can be programmed to treat these inputs as either synchronous or asynchronous inputs by tying its $\overline{\text{ASYNC}}$ input (pin 15) either high or low, respectively. SACK must be treated as asynchronous unless it has been synchronized to the microprocessor's clock with an external flip-flop.

$\overline{\text{SACK}}$ set-up time is shown in Figures 35 and 36, and is calculated in Equation 7. This equation indicates that, at worst case, one wait state will be generated (n = 1). This satisfies the requirements of the system, namely one wait state for reads and zero (or more) wait states for writes.

$\overline{\text{SACK}}$ Set-Up Time Margin (N = 0) (Equation 7.)

$$
\begin{aligned}
&= \overline{\text{RD}} \text{ or } \overline{\text{WR}} \text{ Active to } \overline{\text{SACK}} \text{ Active Delay} \\
&= (N)\text{TCLCL}(5\text{MHz}) + t_{PLH}\text{min}(7404)^{[1]} - \\
&\quad \text{TCLMLmax}(8288) - t_{CA}\text{max}(8203) \\
&\quad - t_{SU}\text{min}(74S74) \\
&= 0 + 1 - 35 - [2(40) + 47] - 3 \\
&= -164 \Rightarrow 1 \text{ wait state wil be generated } (N = 1)
\end{aligned}
$$

We have only looked at "worst case" $\overline{\text{SACK}}$ set-up time so far, to determine the maximum number of wait states that will be generated (assuming no delays due to a refresh cycle in progress). We should look at "best

[1] Not specified — use 1 ns.

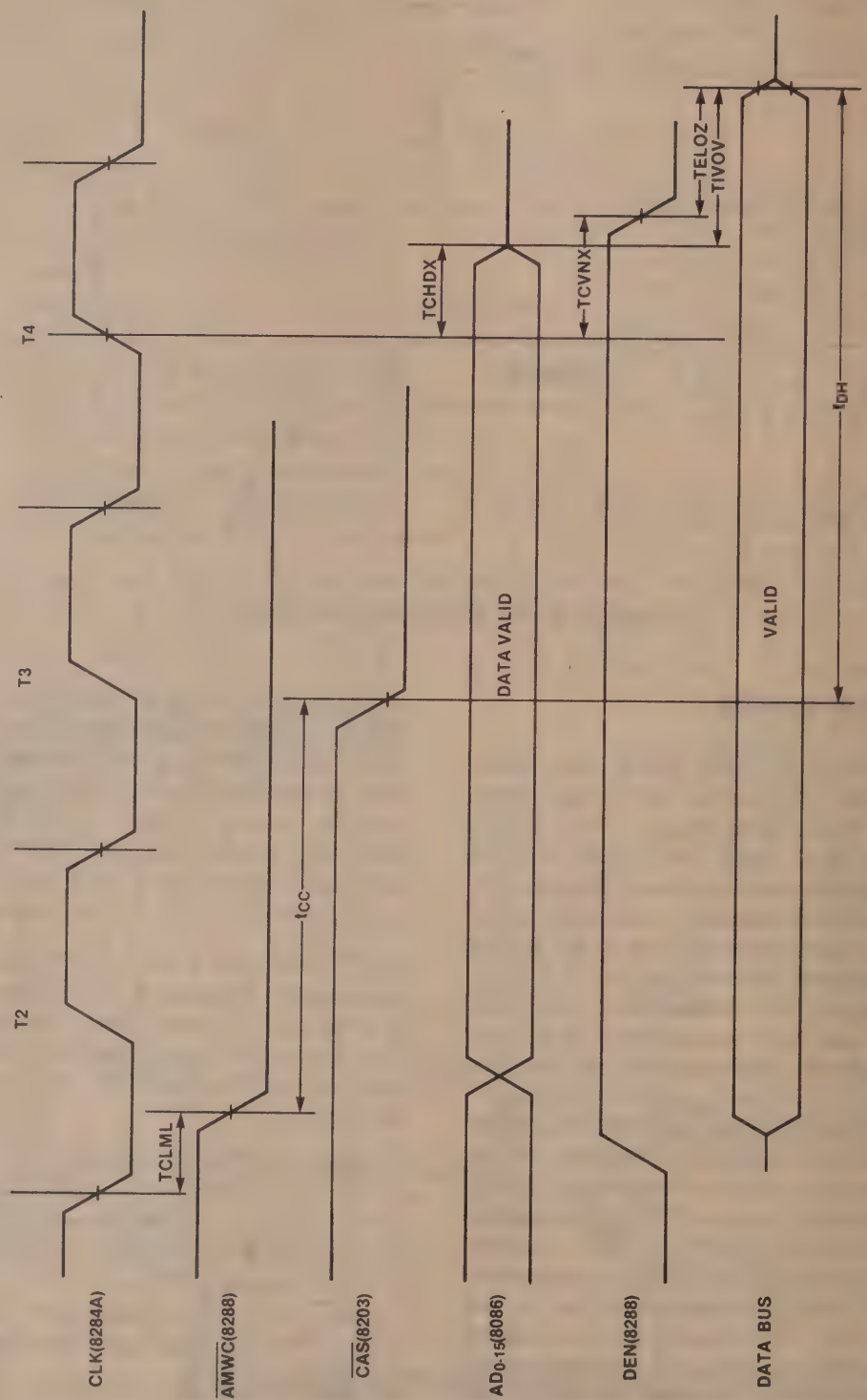


Figure 34. Write Data Hold Time Margin

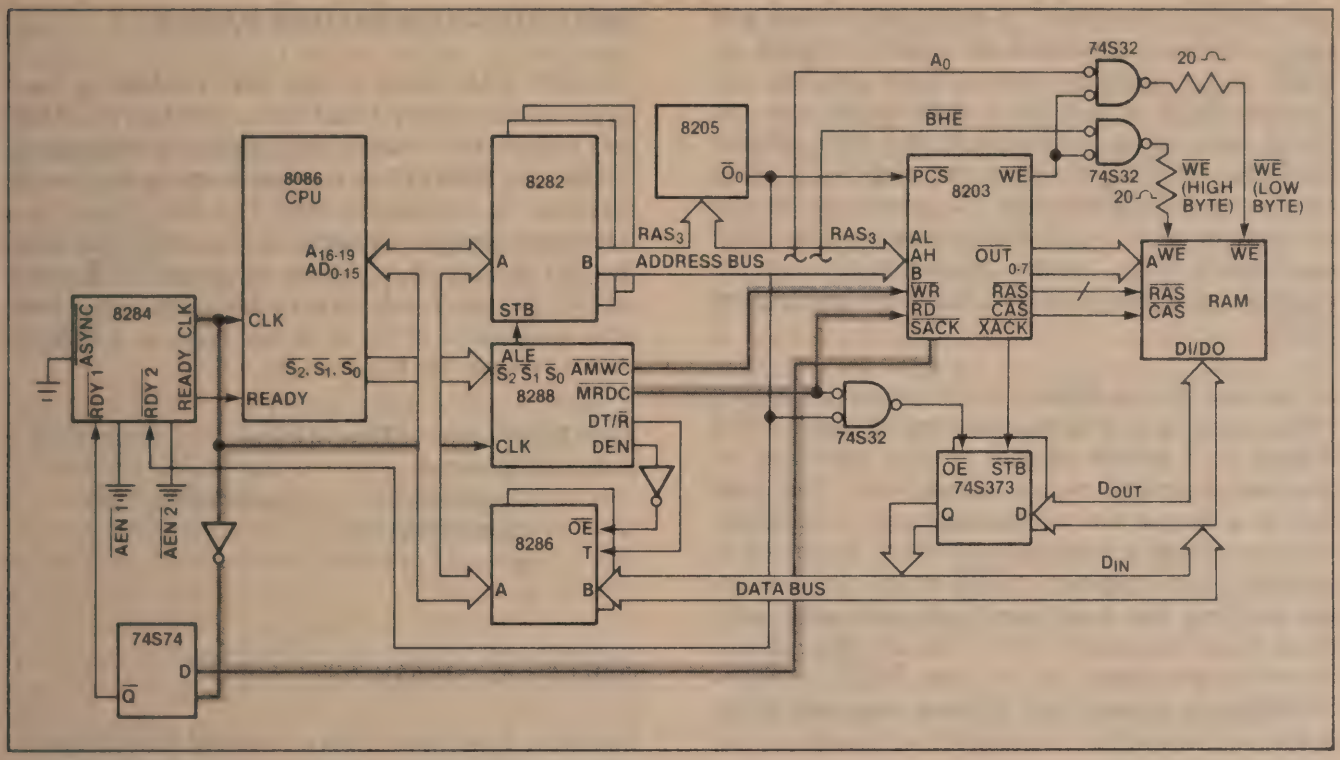


Figure 35. SACK Set-Up Time Margin

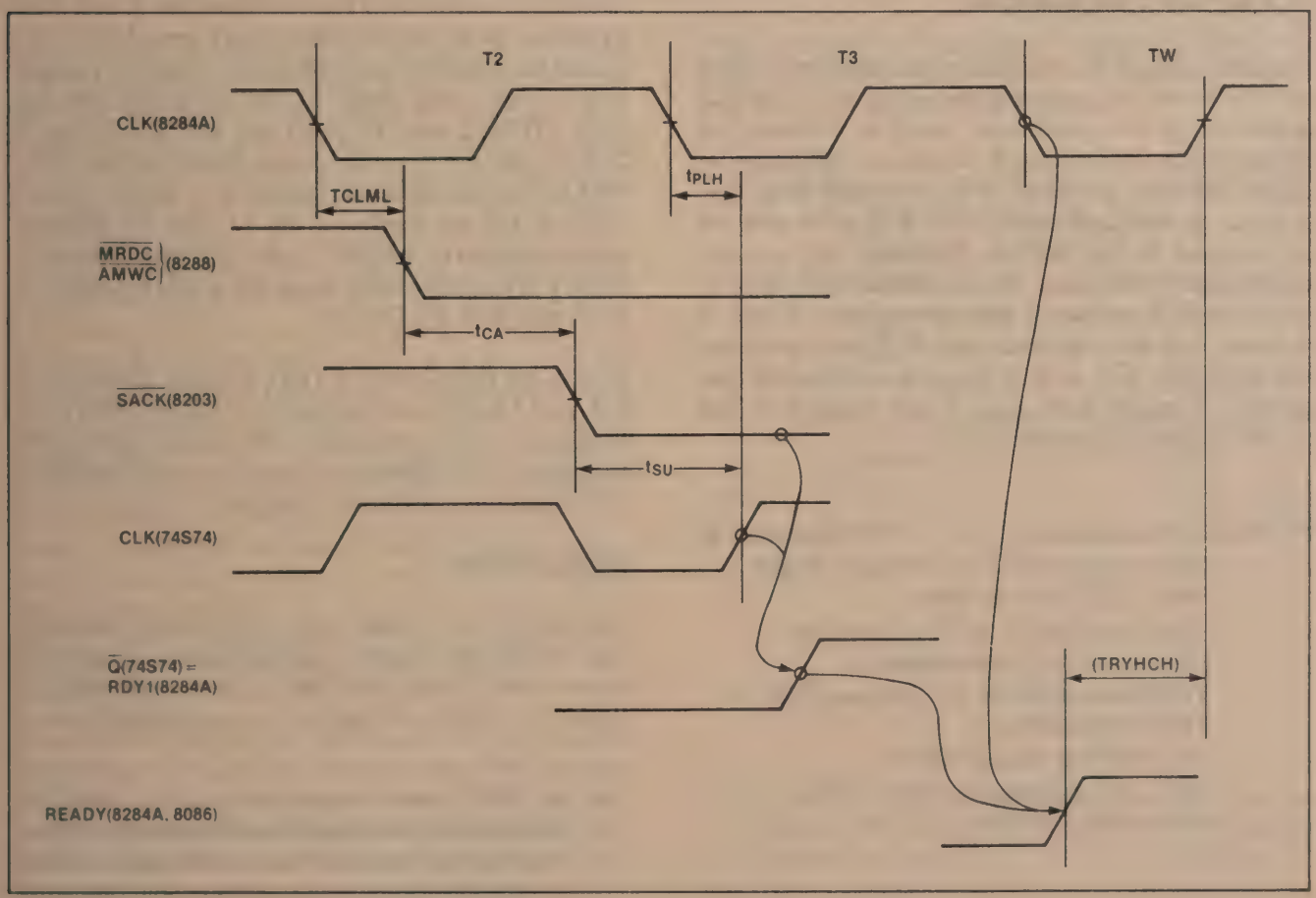


Figure 36. SACK Set-Up Time Margin

case" $\overline{SACK}$ timing also, to make sure enough wait states are always generated. Note that in Figure 35, $\overline{SACK}$ goes through an external 74S74 flip-flop; this samples $\overline{SACK}$ on-half clock cycle earlier than the 8284A does (on the same clock edge that activates $\overline{MRDC}$ or $\overline{AMWC}$), effectively reducing $\overline{SACK}$ set-up time by one-half clock period. This guarantees the proper number of wait state will be generated for "best case" $\overline{SACK}$ timing. Adding this flip-flop does not increase the worst case number of wait states generated by $\overline{SACK}$.

In the case where a memory cycle is requested while a refresh cycle is in progress, the memory cycle will be delayed by a variable amount of time, depending on how near the refresh cycle is to completion. This delay may be as long as one full memory cycle if the refresh was just starting; this time is about 650 ns, depending on the 8203's clock frequency. $\overline{SACK}$ set-up, read data set-up, and write data hold times to the microprocessor's clock are not the same as in the usual case where there is no refresh interference. In this case, $\overline{SACK}$ is delayed until the read or write cycle has been completed by the RAM, so that there is no possibility of terminating the cycle too soon.

PCS SET-UP TIME MARGIN

The 8203's $\overline{RD}$, $\overline{WR}$, and ALE inputs must be qualified by $\overline{PCS}$ in order to perform a memory cycle. If the $\overline{PCS}$ active set-up time parameter (t_{PCS}) is violated, the memory cycle will be delayed. In this case all maximum delays normally measured from command (t_{CR}, t_{CC}, t_{CA}) will be measured instead from $\overline{PCS}$ active and will be increased by t_{PCS} (20 ns). Minimum t_{CR}, t_{CC}, t_{CA} delays remain the same, but are measured from command or $\overline{PCS}$ whichever goes active later. If t_{PCS} is violated, care must be taken that $\overline{PCS}$ does not glitch low while $\overline{RD}$, $\overline{WR}$, or ALE is active, erroneously triggering a memory cycle. t_{PCS} is not violated in this system, however (Equation 8).

$\overline{PCS}$ Set-Up Time Margin (Equation 8.)

= CPU Address Valid to Command Active Delay - $\overline{PCS}$ Decode Time

= TCLCL(5MHz) + TCLMLmin(8288) - [Greater of TCLAVmax(8086) + TIVOVmax(8282) or TCLLHmax(8288) + TSHOVmax(8282)] - $t_p$max(8205) -t_{PCS}min(8203)

= 200 + 10 - [Greater of (110 + 30) or (15 + 45)] - 18 - 20

= 32

RAM DATA OUT HOLD TIME MARGIN

The 8203 CAS output is only held valid for a fixed length of time during a read cycle, after that the RAM data outputs are 3-stated. This time is not long enough to allow the 8086 to read the data from the bus, so the data must be latched externally. This latch should be a transparent type and should be strobed by $\overline{XACK}$ from the 8203. Because the minimum time from $\overline{XACK}$ active to $\overline{CAS}$ inactive is only 10 ns, a latch with a data hold time requirement of 10 ns or less (such as a 74S373) should be used (see Equation 9).

RAM Data Out Hold Time Margin, from $\overline{XACK}$ Active (Equation 9.)

= t_{ACK}min(8203) + t_{OFF}min(2164A - 20) - $t_H$min(74S373)[1]

= 10 + 0 - 10

= 0

OTHER CALCULATIONS

Equations 3, 4, 6 and 7 may be solved directly for N, where N is the number of wait states, to find how many wait states are required at a given frequency. Alternatively, a number may be substituted for N and these equations solved for the 8086's clock period, TCLCL, to find the maximum microprocessor frequency possible with N wait states. Note that the clock high and low times (TCHCL and TCLCH) are also a function of TCLCL. Be sure to use the proper speed selection of the 8086 in this calculation, as various A.C. parameters are different and the result may be different for different speed selections of the 8086, even at the same frequency. Be sure to check the other equations at this frequency to make sure they are OK, too.

Finally, for given values of TCLCL and N, Equations 3, 4, 6, and 7 may be checked to find the lowest 8203 clock frequency which will allow the same system performance, if it is desired to operate at some frequency other than the 25 MHz we assumed.

CONCLUSION

This design will operate with, at worst case, one wait state (except for refresh) at microprocessor frequencies up to 6 MHz, using slow (200 ns access time) dynamic RAMs. At 6 MHz, it is limited by a lack of $\overline{SACK}$ set-up

[1] A 74S373 must be used to meet this timing requirement. Even though worst case margin is 0 ns, this is not a critical timing, as valid data will hold on the latch inputs for a considerable time after the RAM outputs 3-state.

time. At 5 MHz, the 8203 can be operated at any clock frequency from 18.432 MHz to 25 MHz, still with only one wait state.

Example 2. 8086 Alternate Configuration System (5 MHz)

Figure 37 shows another 8086 Max mode system at 5 MHz, but this time using the Alternate Configuration, which allows it to operate with *no wait states* (except for refresh).

The system in the previous example was limited by $\overline{SACK}$ set-up time. $\overline{SACK}$ set-up time can be improved by sampling $\overline{SACK}$ later; this has been done by changing the clock edge used to sample $\overline{SACK}$, allowing roughly ⅔ clock period longer. $\overline{SACK}$ set-up time (and read data access time and write data hold time) margin can also be improved by activating the $\overline{RD}$ or $\overline{WR}$ inputs of the 8203 earlier in the 8086's bus cycle; this is the purpose of the extra logic in Figure 37 (I.C.s A8 - A11). These generate advanced $\overline{RD}$ and $\overline{WR}$ signals timed from the falling edge of ALE, which occurs roughly ⅓ clock period sooner than the $\overline{MRDC}$ and $\overline{AMWC}$ are generated by the 8288 Bus Controller. Altogether, these changes allow about one 8086 clock period more set-up time for $\overline{SACK}$.

Let's look at this logic in more detail. An Intel 8205 (A8) is used to decode the 8086's status outputs $\overline{S_{0\text{-}2}}$. An opcode fetch, memory read, or memory write decode to 8205 outputs 4, 5, and 6, respectively. These outputs go to the D inputs of two 74S74 flip-flops. The Q output of flip-flop A10.2 is an advanced memory read signal and the Q output of A11.2 is an advanced memory write signal. As shown in Figure 37, the 8203 is not activated for opcode fetches, but it can be if 8205 outputs 4 and 5 are ORed with the unused 74S00 gate (A9.4) and the $\overline{Q}$ output of A10.2 used instead of Q. Both flip-flops are clocked by the falling edge of ALE to generate the advanced commands. Flip-flop A10.1 is clocked by the trailing edge of either $\overline{AMWC}$ (Advanced Memory Write Command) or $\overline{MRDC}$ (Memory Read Command) from the 8288 bus controller (A6), indicating that the 8086 has completed the memory cycle. A10.1, in turn, presets both the A10.2 and A11.2 flip-flops to terminate the advanced memory read and write signals to the 8202A. A10.1 is then preset to its initial state by ALE going active at the start of the next bus cycle.

Because RAM write cycles are started very early in the 8086's bus cycle using this logic, the 8203 will activate $\overline{CAS}$ to the RAMs (latching write data) before the data is valid from the 8086. This requires delaying $\overline{WE}$ to the RAMs and performing a "late write" (explained earlier under *Dynamic RAMs*) in order to allow more time for the write data to arrive. But the $\overline{WE}$ signal must not be delayed so long that there is no longer enough data hold time, measured from when $\overline{WE}$ goes active; or that the $\overline{WE}$ active to $\overline{CAS}$ inactive delay spec or the RAM (t_{RWL}) is violated. None of the control signals from the 8086 or 8288 bus controller satisfy both of these timing constraints, so such a signal is generated by flip-flop A11.1, which serves to delay $\overline{AMWC}$ from the bus controller by an amount of time equal to TCLCH (the low time of the 8086's clock). A11.1 is also preset by A10.1 at the end of the memory cycle. The Q output of A11.1 is ANDed with $\overline{WE}$ from the 8203 by A14.1 to form a delayed RAM $\overline{WE}$. As in the previous example, this signal is then ANDed with $\overline{BHE}$ and AO to form the $\overline{WE}$ for the high and low bytes of RAM, respectively.

A total of four packages (three 14-pin and one 16-pin) of TTL logic are required.

The dynamic RAM interface timings are identical to the last example (Equations 1 (a)-(h)); 2164A-20 RAMs will be used again.

ADDRESS SET-UP AND HOLD TIME MARGINS

Address set-up and hold time margins are given in Equations 10 and 11, respectively. An 8086-*2* microprocessor has been used instead of the standard 8086, as this speed-selected part gives better address set-up to $\overline{RD}$ or $\overline{WR}$ times, which this design needs since it uses advanced $\overline{RD}$ and $\overline{WR}$ commands.

Row Address Set-Up Time Margin[1] (Equation 10.)

= CPU Address to Adv. $\overline{RD}$ Delay + $\overline{RAS}$ Delay - Address Delays

= TCLCHmin(8284A) + TCHLLmin(8288)[2] + t_{PLH}min(74S00)[3] + t_{PHL}min(74S74)[2] + t_{CR}min(8203) - [Greater of TCLAVmax(8086 - 2) + TIVOVmax(8282) or TCLLHmax(8288) + TSHOVmax(8282)] - t_{AD}max(8203) - t_{ASR}min(2164A-20)

= [⅔(200) - 15] + 2 + 1 + 2 + [(40) + 30] - [Greater of (60 + 30) or (15 + 45)] - 40 - 0

= 63

[1] Read or write cycles only. Eq. 1b gives this timing for refresh cycles.

[2] Not specified — use 2 ns.

[3] Not specified — use 1 ns.

Address Hold Time Margin (N = 0) (Equation 11.)

= CPU Address Hold Time from Adv. $\overline{RD}$ Active - $\overline{RAS}$ Inactive Delays

= (3+N)TCLCL(5MHz) + TCHCLmin(8284A) + TCLLHmin (8288) + TSHOVmin(8282) - TCLMLmax(8288) - t_{CC}max(8203) - t_{RSH}max(8203)

= (3)200 + [⅓(200) + 2] + 2 + 5 - 35 - [4(40) + 85] - [5(40) + 20]

= 175

READ DATA ACCESS TIME MARGIN

Read data access time margin is shown in Equation 12; no wait states are required for read cycles, even with 200 ns access time RAMs.

Read Data Access Time Margin (N = 0) (Equation 12.)

= Adv. $\overline{RD}$ to Data Valid Delay - $\overline{CAS}$ Delay - Read Data Delays

= (2+N)TCLCL(5MHz) + TCHCLmin(8284A) - TCHLLmax(8288) - t_{PLH}max(74S00) - t_{PHL}max(74S74) - t_{CC}max(8203) - t_{CAC}max(2164A—20) - $t_p$max(74S373) - TIVOVmax(8286) - TDVCLmin(8086—2)

= (2)200 + [⅓(200) + 2] - 15 - 5 - 10 - [4(40) + 85] - 110 - 30 - 30 - 20

= 3

WRITE DATA SET-UP AND HOLD TIME MARGINS

Write data set-up and hold times are shown in Equations 13 and 14, respectively. No wait states are required during write cycles. Note that write data set-up has been guaranteed by delaying $\overline{WE}$ from the 8203 with clocked $\overline{AMWC}$ from the bus controller and performing "late write" cycles; write data set-up time would not be satisfied otherwise. Equation 15 verifies that $\overline{WE}$ has not been delayed too long to meet the RAM's $\overline{WE}$ active to $\overline{RAS}$ inactive set-up time (t_{RWL}). The RAM's $\overline{WE}$ active to $\overline{CAS}$ inactive set-up time (t_{CWL}) is also satisfied, since $\overline{CAS}$ does not go inactive until at least 20 ns after $\overline{RAS}$.

Write Data Set-Up Time Margin (Equation 13.)

= CPU Data to Clocked $\overline{AMWC}$ Set-Up + $\overline{WE}$ Delays - Data Delays

= TCLCHmin(8284A) + t_{PHL}min(74S74)[1] + (2)t_{PHL}min(74S32)[1] - TCLDVmax(8086-2) - TIVOVmax(8286) - t_{DS}min(2164A—20)

= [⅔(200) - 15] + 2 + (2)2 - 60 - 30 - 0

= 34

Write Data Hold Time Margin (N = 0) (Equation 14.)

= CPU Data Hold Time from Clocked $\overline{AMWC}$ + Data Delays - $\overline{WE}$ Delays

= (2 + N)TCLCL(5MHz)

= TCHDXmin(8086—2) + TIVOVmin(8286) - t_{PHL}max(74S74) - (2)t_{PHL}max(74S32) - t_{DH}min(2164A—20)

= (2)200 + 10 + 5 - 10 - (2)7 - 45

= 346

$\overline{WE}$ Active Set-Up Time Margin to $\overline{RAS}$ Inactive (Equation 15.)

= TCHLLmin(8284A)[1] + t_{PLH}min(74S00)[2] + t_{CC}min(8203) + t_{RSH}min(8203) - t_{SKEW}(74S74)[3] -(2)t_{PHL}max(74S32) - t_{RWL}min(2164A—20) - TCLCL(5MHz)

= 2 + 1 + [3(40) + 25] + [5(40) - 30] - 2 - (2)7 - 50 - 200

= 52

$\overline{SACK}$ SET-UP TIME MARGIN

Equation 16 shows that $\overline{SACK}$ set-up time is satisfied; no wait states will be generated for read or write cycles (except for refresh).

$\overline{SACK}$ Set-Up Time Margin (N = 0) (Equation 16.)

= (1 + N)TCLCL(5MHz) - TCHLLmax(8288) - t_{PLH}max(74S00) - t_{PHL}max(74S74) - t_{CA}max(8203) - t_{SU}min(74S74)

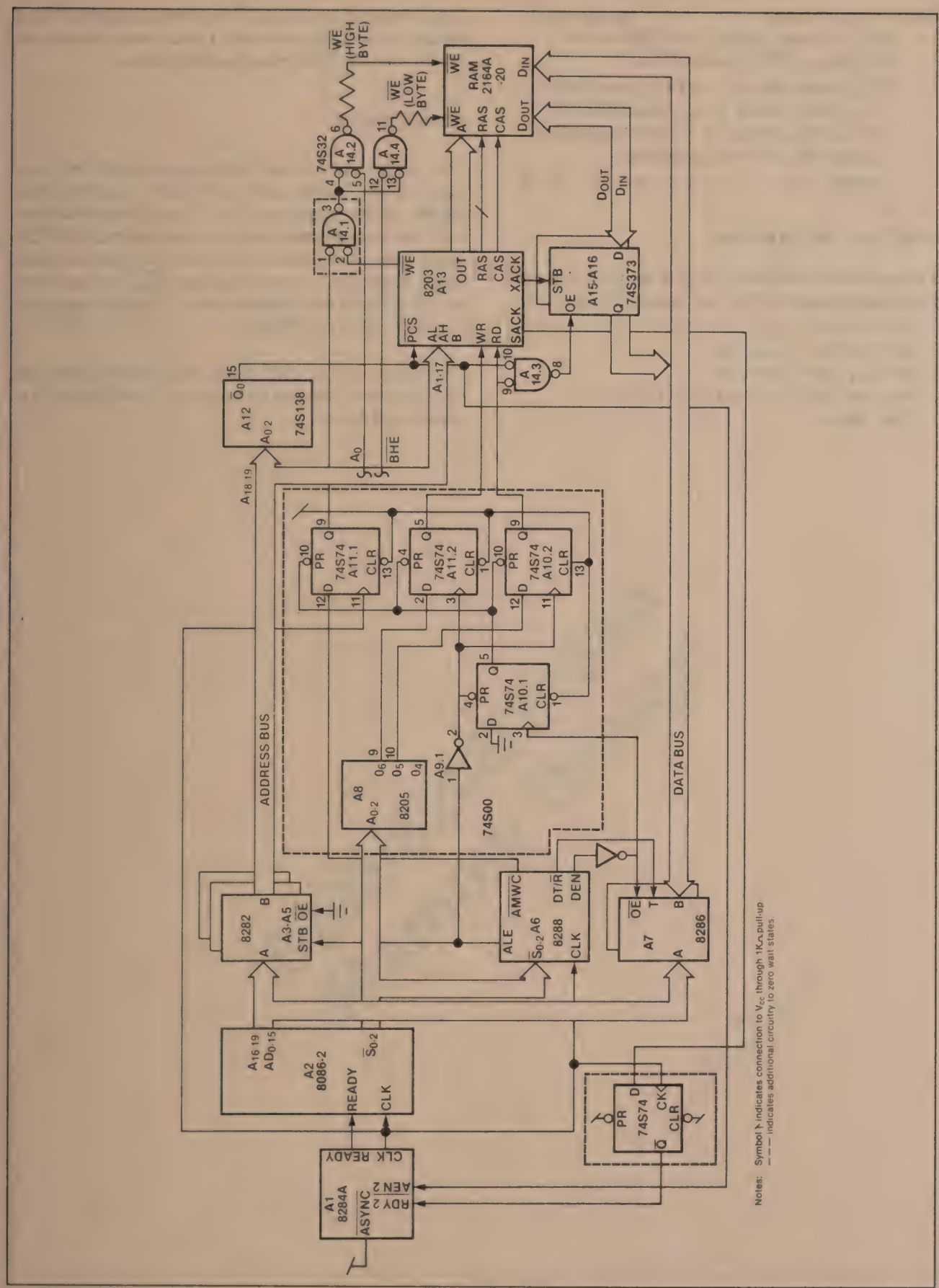
= 200 - 35 - 5 - 10 [2(40) + 47] - 3

= 20

[1] Not specified — use 2 ns.

[2] Not specified — use 1 ns.

[3] t_{SKEW}(74S74) is max. skew between t_{PHL}(Q output, from CLK) of two Q outputs in same package — use = 2 ns.



Notes: Symbol ⊢ indicates connection to V_{CC} through 1KΩ pull-up
– – – indicates additional circuitry to zero wait states.

Figure 37. 8086 Alternate Configuration System

$\overline{PCS}$ Set-Up Time Margin (Equation 17.)

= CPU Address Valid to Adv. $\overline{RD}$ or Adv. $\overline{WR}$ Delay - $\overline{PCS}$ Decode Time

= TCLCHmin(8284A) + TCHLLmin(8288)[1] + t_{PLH}min(74S00) + t_{PHL}min(74S74)[1] - TCLAVmax(8086—2) - TIVOVmax(8282) - $t_p$max(74S138[3] - t_{PCS}min(8203)

= [⅔(200) - 15] + 2 + 1 + 2 - 60 - 30 - 12 - 20

= 1

PCS SET-UP TIME MARGIN

$\overline{PCS}$ set-up time for the 8203 (t_{PCS}) is satisfied, but not with as much margin in the last example (Figure 17).

[1] Not specified — use 2 ns.
[2] Not specified — use 1 ns.
[3] Must use 74S138 to maintain $\overline{PCS}$ Set-Up Time Margin.

This is because the $\overline{RD}$ and $\overline{WR}$ commands are activated earlier in the microprocessor's bus cycle, leaving less time to decode $\overline{PCS}$ from the address bus.

CONCLUSION

This design will operate with a guaranteed zero wait states up to 5 MHz using slow (200 ns access time) RAMs. At this frequency, it is limited by both read and write data set-up times, and to a lesser extent, by $\overline{SACK}$ set-up time. Using faster RAMs will not raise the maximum frequency, as write data and $\overline{SACK}$ set-up times are not affected by the RAM speed. The 8203 operating frequency must be 25 MHz.

This design can be used (with some modifications) to allow one wait state performance up to 8086 clock frequency of 8 MHz.

intel

APPLICATION NOTE

AP-141

October 1981

8203/8206/2164A Memory Design

Brad May
Peripherals Applications

order number:210315.001

8203/8206/2164A Memory Design

Contents

ABSTRACT

DESIGN

CONCLUSION

ABSTRACT

This Application Note shows an error corrected dynamic RAM memory design using the 8203 64K Dynamic RAM Controller, 8206 Error Detection and Correction Unit and 150 ns 64K Dynamic RAMs with a minimum of additional logic.

The goals of this design are to:

1. Control 128K words × 16 bits (256 KB) of 64K dynamic RAM.
2. Support 150 ns dynamic RAMs.
3. Write corrected data back into dynamic RAM when errors are detected during read operations.
4. To use a minimum of additional logic.

It is not the goal of this design to:

1. Provide the maximum possible performance.
2. Provide features like error logging, automatic error scrubbing and dynamic RAM initialization on power-up, or diagnostics, although these features can be added.

DESIGN

Figure 1 shows a memory design using the 8206 with Intel's 8203 64K Dynamic RAM Controller and 150 ns 64K Dynamic RAMs. As few as three additional ICs complete the memory control function (Figure 2).

For simplicity, all memory cycles are implemented as single-cycle read-modify-writes, shown in Figure 3. This cycle differs from a normal read or write primarily when the dynamic RAM write enable ($\overline{WE}$) is activated. In a normal write cycle, $\overline{WE}$ is activated early in the cycle; in a read cycle, $\overline{WE}$ is inactive. A read-modify-write cycle consists of two phases. In the first phase, $\overline{WE}$ is inactive, and data is read from the dynamic RAM; for the second phase, $\overline{WE}$ is activated and the (modified) data is written into the same word in the dynamic RAM. Dynamic RAMs have separate data input and output pins so that modified data may be written, even as the original data is being read. Therefore data may be read and written in only one memory cycle.

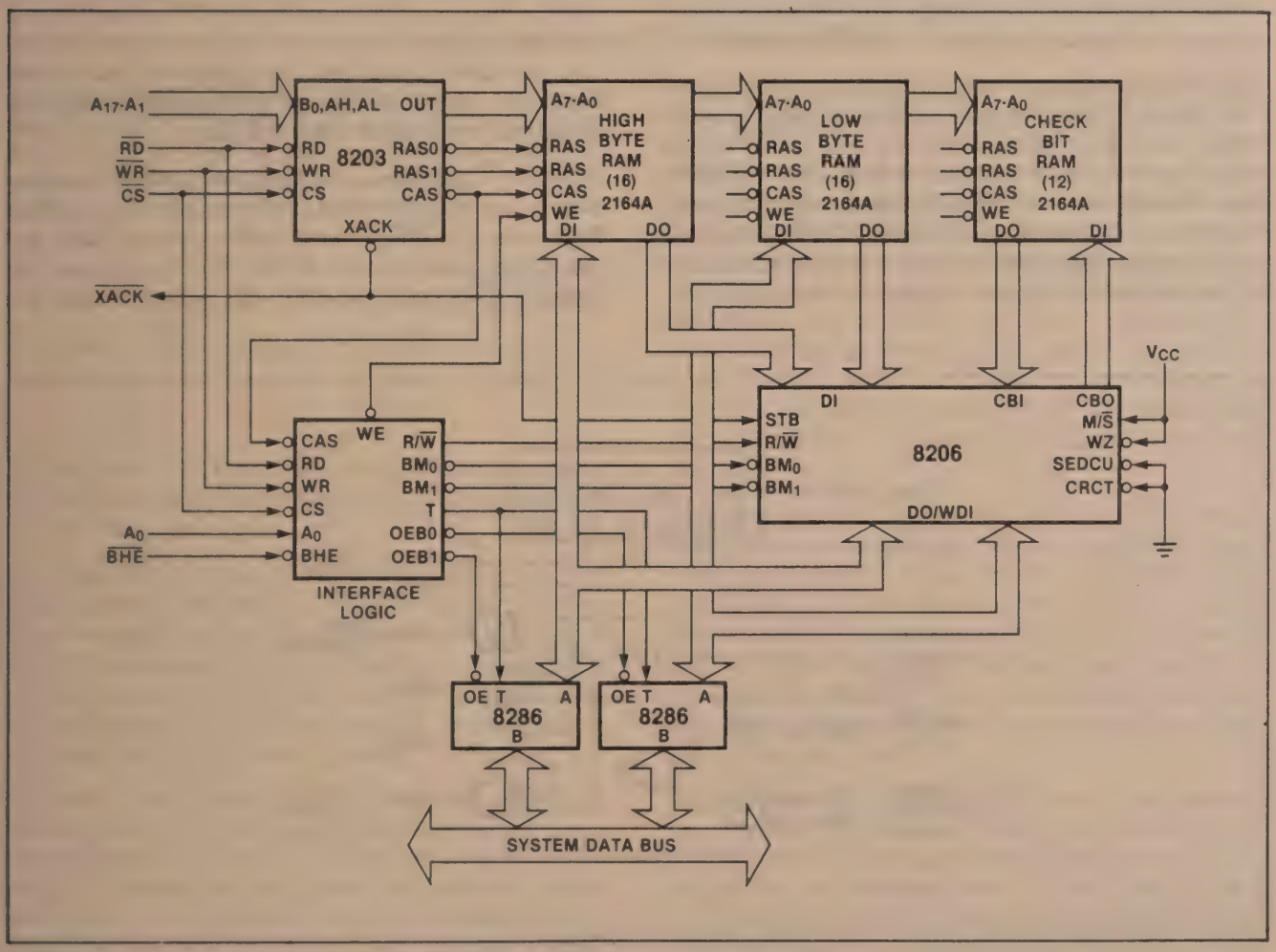


Figure 1. 8203/8206 Memory System

In order to do read-modify-writes in one cycle, the dynamic RAM's $\overline{CAS}$ strobe must be active long enough for the 8206 to access data from the dynamic RAM, correct it, and write the corrected data back into the dynamic RAM. $\overline{CAS}$ active time is an 8203 spec (t_{CAS}), and is dependent on the 8203's clock frequency. The clock frequency and dynamic RAM must be chosen to satisfy Equation 1.

(Eq. 1)

8203		Dynamic RAM		8206		8206		Dynamic RAM		Dynamic RAM
t_{CAS}min	≥	t_{CAC}	+	TDVQV	+	TQVQV	+	t_{DS}	+	t_{CWL}
5(54)−10	≥	85	+	67	+	59	+	0	+	40
260	≥	251		✓						

The 8203 itself performs normal reads and writes. In order to perform read-modify-writes, all that is needed is to change the timing of the $\overline{WE}$ signal. In this design, $\overline{WE}$ is generated by the interface logic in Figure 2—the 8203 $\overline{WE}$ output is not used. All other dynamic RAM control signals come from the 8203. A 20-ohm damping resistor is used to reduce ringing of the WE signal. These resistors are included on-chip for all 8203 outputs.

The interface logic generates the R/$\overline{W}$ input to the 8206. This signal is high for read cycles and low for write cycles. During a read-modify-write cycle, R/$\overline{W}$ is first high, then low. The falling edge of R/$\overline{W}$ tells the 8206 to latch its syndrome bits internally and generate corrected check bits to be written to dynamic RAM. Corrected data is already available from the DO pins. No control signals at all are required to generate corrected data.

R/$\overline{W}$ is generated by delaying $\overline{CAS}$ from the 8203 with a TTL-buffered delay line. This allows the 8206 sufficient time to generate the syndrome; this delay, $t_{DELAY\ 1}$, must satisfy Equation 2.

(Eq. 2)

		Dynamic RAM		8206
$t_{DELAY\ 1}$	≥	t_{CAC}	+	TDVRL
150	≥	85	+	34
150	≥	119	✓	

The 8206 uses multiplexed pins to output first the syndrome word and then check bits. This same R/$\overline{W}$ signal may be used to latch the syndrome word externally for error logging. The 8206 also supplies two useful error signals. $\overline{ERROR}$ signals the presence of an error in the data or check bits. CE tells if the error is correctable (single bit in error) or uncorrectable (multiple bits in error).

In the event that an uncorrectable error is detected, the 8206 will force the Correctable Error (CE) flag low; this may be used as an interrupt to the CPU to halt execution and/or perform an error service routine. In this case the 8206 outputs data and check bits just as they were read, so that the data in the dynamic RAM is left unaltered, and may be inspected later.

After R/$\overline{W}$ goes low, sufficient time is allowed for the 8206 to generate corrected check bits, then the interface logic activates $\overline{WE}$ to write both corrected data and check bits into dynamic RAM. $\overline{WE}$ is generated by delaying CAS from the 8203 with the same delay line

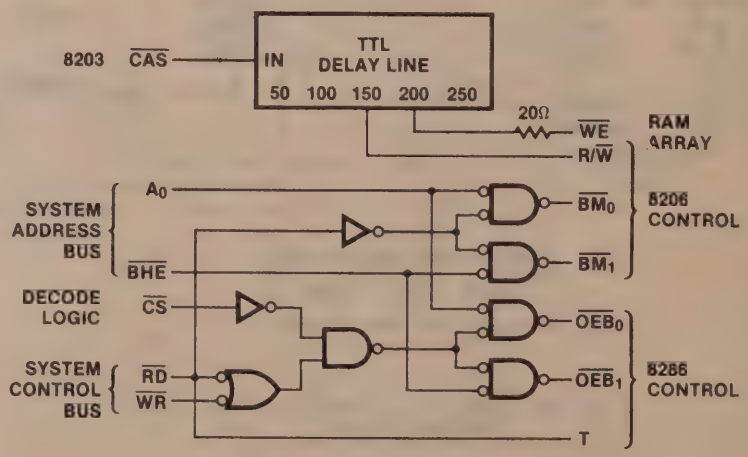


Figure 2. Interface Logic

210315-001

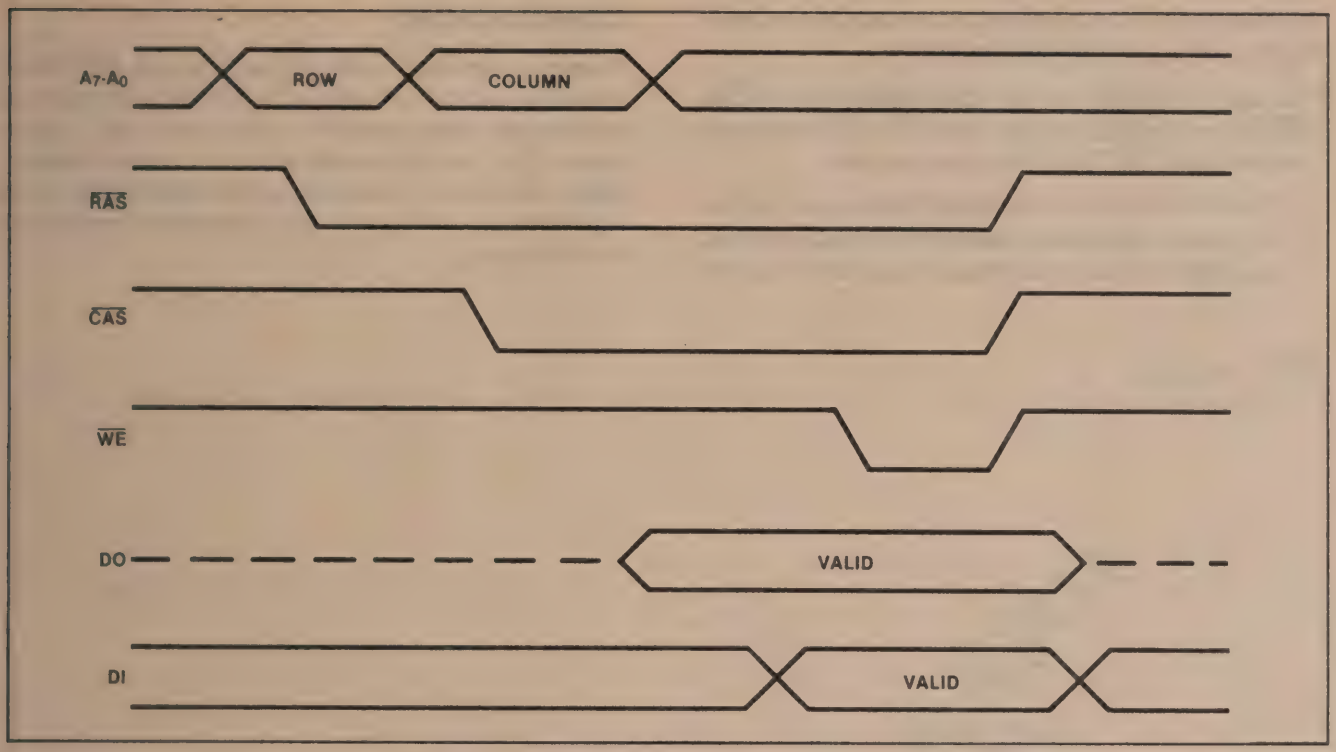


Figure 3. Single-Cycle Read-Modify-Write

used to generate R/$\overline{W}$. This delay, $t_{DELAY\ 2}$, must be long enough to allow the 8206 to generate valid check bits, but not so long that the t_{CWL} spec of the RAM is violated. This is expressed by Equation 3.

(Eq. 3)

$$t_{DELAY\ 1} + \underbrace{TRVSV}_{8206} \le t_{DELAY\ 2} \le \underbrace{t_{CAS}min}_{8203} - \underbrace{t_{CWL}}_{\text{Dynamic RAM}}$$

$$150 + 42 \le 200 \le 260 - 40$$

$$192 \le 200 \le 220 \quad \checkmark$$

Unlike other EDC chips, errors in both data *and check bits* are automatically corrected, without programming the chip to a special mode.

Since the 8203 terminates $\overline{CAS}$ to the dynamic RAMs a fixed length of time after the start of a memory cycle, a latch is usually needed to maintain data on the bus until the 8086 completes the read cycle. This is conveniently done by connecting $\overline{XACK}$ from the 8203 to the STB input of the 8206. This latches the read data and check bits using the 8206's internal latches.

The 8086, like all 16-bit microprocessors, is capable of reading and writing single byte data to memory. Since the Hamming code works only on entire words, if you want to write one byte of the word, you have to read the entire word to be modified, do error correction on it, merge the new byte into the old word inside the 8206, generate check bits for the new word, and write the whole word plus check bits into dynamic RAM. A byte write is implemented as a Read-Modify-Write.

Why bother with error correction on the old word? Suppose a bit error had occurred in the half of the old word not to be changed. This old byte would be combined with the new byte, and new check bits would be generated for the whole word, *including the bit in error*. So the bit error now becomes "legitimate"; no error will be detected when this word is read, and the system will crash. You can see why it is important to eliminate this bit error before new check bits are generated. Byte writes are difficult with most EDC chips, but easy with the 8206.

Referring again to Figure 2, the 8206 byte mark inputs ($\overline{BM_0}$, $\overline{BM_1}$), are generated from A0 and $\overline{BHE}$, respectively, of the 8086's address bus, to tell the 8206 which byte is being written. The 8206 performs error correction on the entire word to be modified, but tri-states its DO/WDI pins for the byte to be written; this byte is provided from the data bus by enabling the corresponding 8286 transceiver. The 8206 then generates check bits for the new word.

During a read cycle, $\overline{BM_0}$ and $\overline{BM_1}$ are forced inactive, i.e., the 8206 outputs both bytes even if 8086 is only reading one. This is done since all cycles are implemented as read-modify-writes, so both bytes of data (plus check bits) must be present at the dynamic RAM data input pins to be rewritten during the second phase of the read-modify-write. Only those bytes actually be-

ing read by the 8086 are driven on the data bus by enabling the corresponding 8286 transceiver.

The output enables of the 8286 transceivers ($\overline{OEB0}$, $\overline{OEB1}$) are qualified by the 8086 $\overline{RD}$, $\overline{WR}$ commands and the 8203 $\overline{CS}$. This serves two purposes:

1. It prevents data bus contention during read cycles.
2. It prevents contention between the transceivers and the 8206 DO pins at the beginning of a write cycle.

CONCLUSION

Thanks to the use of a 68-pin package, the 8206 Error Detection and Correction Unit is able to implement an architecture with separate 16 pin input and output busses. The resulting simplification of control requirements allows error correction to be easily added to an 8203 memory subsystem with a minimal amount of interface logic.

intel®

APPLICATION NOTE

AP-167

August 1983

Interfacing the 8207 Dynamic RAM Controller to the iAPX 186

JIM SLEEZER
APPLICATION ENGINEER

©Intel Corporation, 1983.

ORDER NUMBER: 230809-001
NOVEMBER 1983

INTRODUCTION

Most microprocessor based workstation designs today use large amounts of DRAM for program storage. A drawback to DRAMs is the many critical timings that must be met. This control function could easily equal the area of the DRAM array if implemented with discrete logic.

The VLSI 8207 Advanced Dynamic RAM Controller (ADRC) performs complete DRAM timing and control. This includes the normal RAM 8 warm-up cycles, various refresh cycles and frequencies, address multiplexing, and address strobe timings. The 8207's system interface and RAM timing and control are programmable to permit it to be used in most applications.

Integrating all of the above functions (plus a dual port and error correcting interfaces) allows the user to realize significant cost savings over discrete logic. For example, comparing the 8207 to the iSBC012B 512K byte RAM board (where the DRAM control is done entirely with TTL), an 8207 design saved board space (3 in^2 vs 10 in^2); required less power (420 ma vs 1220 ma); and generated less heat. Moreover, design time was reduced, and increased margins were achieved due to less skewing of critical timings. This comparison is based on a single port design and did not include the 8207's RAM warm-up, dual-port and error correcting features. If these features were fully implemented, there would be no change to the 8207 figures, listed above, while the TTL figures would easily double.

This Application Note will illustrate an iAPX design with the 8207 controlling the dynamic RAM array. The reader should be familiar with the 8207 data sheet, the 80186 data sheet, and a RAM data sheet*.

DESIGN GOALS

The main objective of this design is for the 80186 to run with no wait states with a Dynamic RAM array. The design uses one port of the 8207. The dual port and error correcting interfaces of the 8207 are covered in separate Application Notes.

The size of the RAM array is 4 banks of 64k RAMs or 512k bytes. The memory is to be interfaced locally to the 80186.

USING THE 8207

The three areas to be considered when designing in the 8207 are:

- 8207 programming logic
- Microprocessor interface
- RAM array

8207 Programming

The 8207 requires up to two 74LS165 shift registers for programming. This design needs one 8 bit shift register, as shown in Figure 1. The 16 bits in the Program Data Word are set as shown in Figure 2. Refresh is done internally, so the REFRQ input must be tied high. The memory commands are iAPX 86 status, so

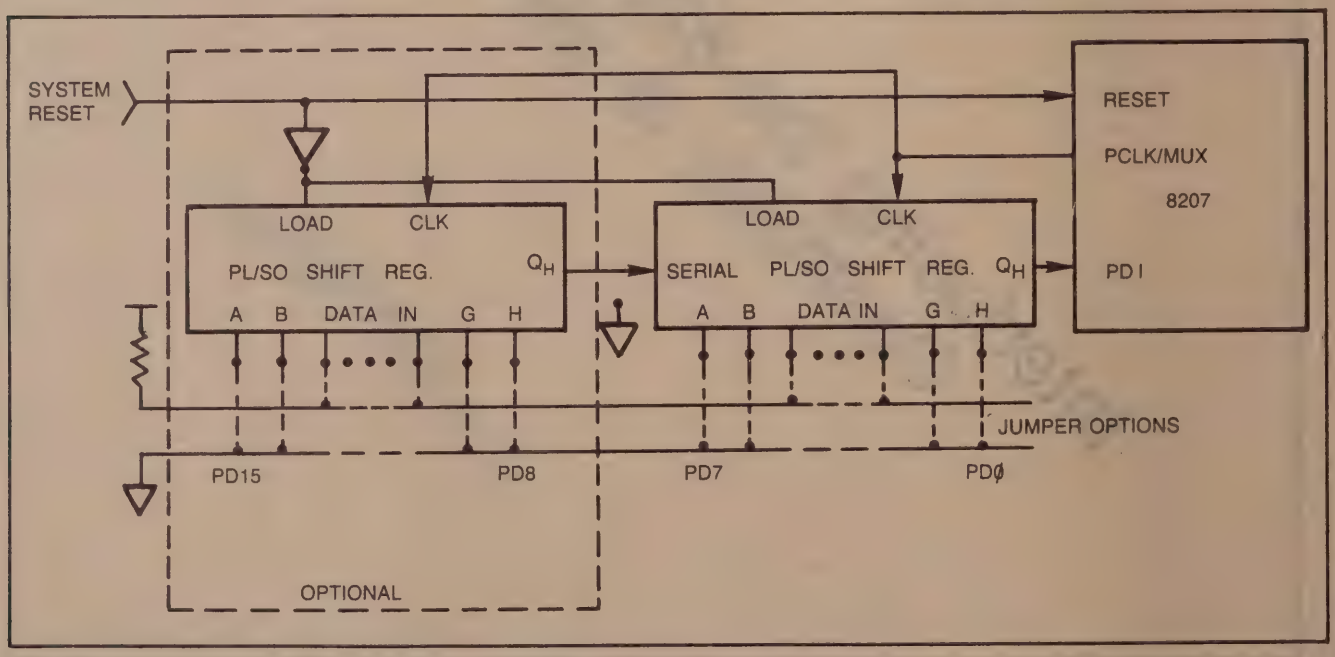
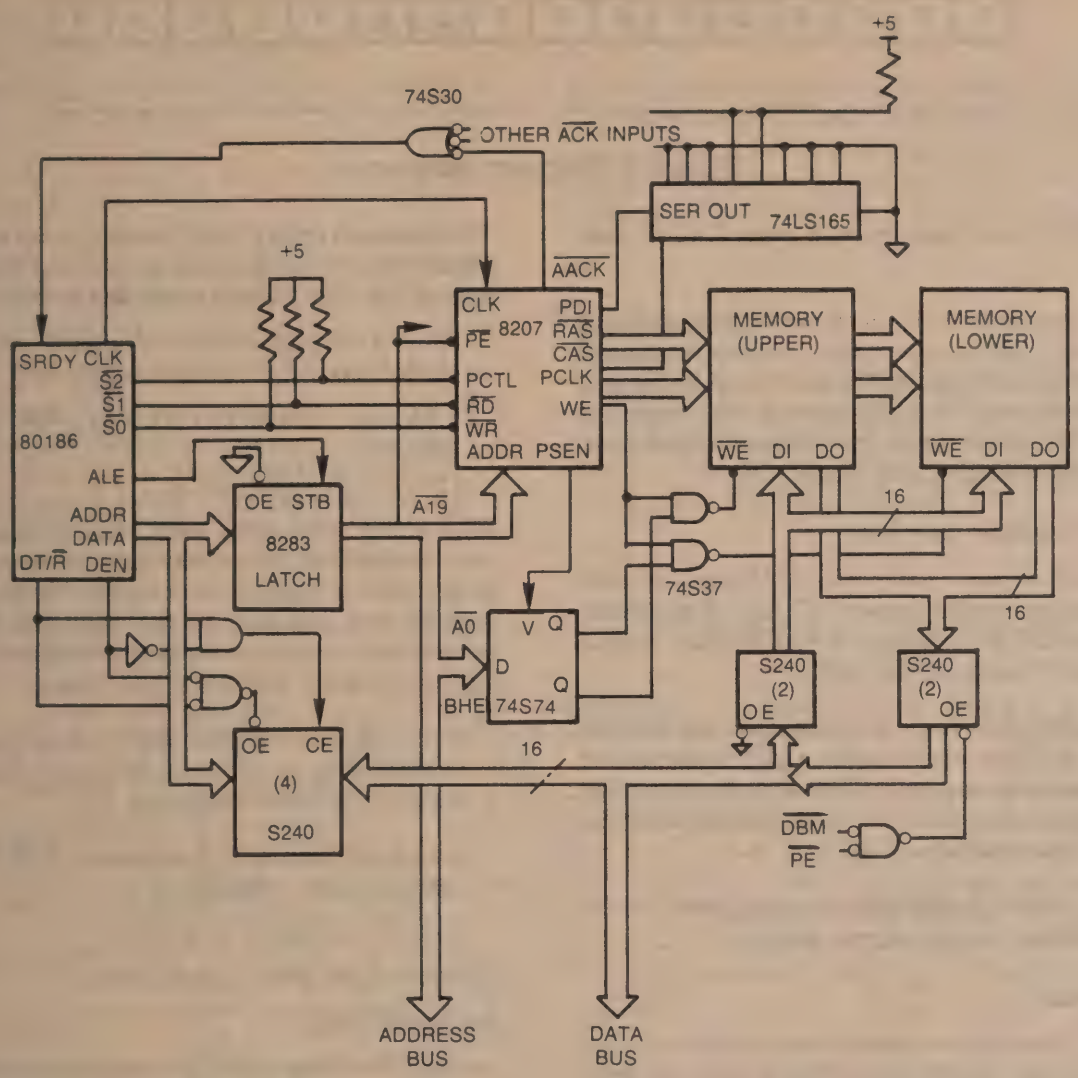


Figure 1. 8207 programming shift registers

*All RAM references in this Application Note are based on Intel's 2164A 64k Dynamic RAM.

230809-001



NOTE: THE 8207 REQUIRES SERIES RESISTORS ON ALL OUTPUTS TO RAM.

Figure 3. 80186 to 8207, non-ECC, synchronous system single port.

the timing of EAACK will always guarantee 2 clocks of address hold time from RAS.

Acknowledge Setup Time

The margin between the 8207 issuing EAACK and the 80186 ready input for no wait states minus delays from clock edges, logic delays, and setup time is calculated as follows.

1 clock - 8207 TCLAKL max -74S30 tPLH @ 15 pf - 80186 TSRYCL ⩾ 0

125 ns - 35 - 22 - 35 = 33 ns

Read Access Margin

The 8207 starts a memory cycle on the falling clock edge between the 80186's T1 and T2. Data must be valid within 2 clocks. Valid data from the RAMs is based upon the CAS access period minus buffer, clock, setup requirements.

2 TCLCL - 8207 TCLCSL @ 150 pf (t34) - DRAM tCAC - 74S240 propagation delay @ 50 pf - additional bus loading delay (250 pf)(1) - 74S240 delay @ 50 pf - 80186 TDVCL ⩾ 0

250 ns - 122 - 85 - 7 - 7 - 7 - 20 = 2 ns

Write Data Setup and Hold Margin

Data from the processor must be valid when WE is issued by the 8207 to meet the RAM specification tDS (2164A = 0 ns), and then held for a minimum of 30 ns.

(1) 74STTL logic derated by .05 ns/pf. 74STTL buffers (240, 37) derated by .025 ns/pf.

0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0

15 … 8 | 7 … 0

Figure 2. Program data word

the PCTLA input must be high when RESET goes inactive.

The differential reset circuit shown in the Data Sheet is necessary only to ensure that memory commands are not received by the 8207 when Port A is changed from synchronous to asynchronous (vice versa for Port B). This design keeps Port A synchronous so no differential reset circuit is needed.

Microprocessor Interface

To achieve no wait states, the 8207 must connect directly to the microprocessor's CLKOUT and status lines. The 8207 Acknowledge (EAACK) must connect to the SRDY input of the 80186.

When the 80186 is reset, it tristates the status lines. The 8207 PCTLA input requires a high to decode the proper memory commands. This is accomplished by using a pull-up resistor or some component that incorporates a pull-up on S2.

The 8207 address inputs are connected directly to the latched/demultiplexed address bus.

RAM Array

The 8207 provides complete control of all RAM timings, warm up cycles, and refresh cycles. All write cycles are "late writes." During write cycles, the data out lines go active. This requires separate data in/out lines in the RAM array.

To operate the 80186 with no wait states, it is necessary to chose sufficiently fast DRAMs. The 150 ns version of the 2164A allows operating the 80186 at 8 MHz, and the 200 ns version up to 7 MHz.

HARDWARE DESIGN

Figure 3 shows a block diagram of the design, and Figure 4 is a timing diagram showing the relationship between the 8207 and the 80186.

8207 Command Setup

Two events must occur for a command to be recognized by the 8207. The 80186 status outputs are sampled by a rising clock edge and Port Enable (PE) is sampled by the next falling clock edge (refer to the Data Sheet wave forms).

The command timing is determined by the period between the status being issued and the first rising clock edge of the 8207, minus setup and delays.

80186 status valid to 8207 rising clock – status from clock delay – 8207 setup to clock ⩾ 0

1 TCLCL – 80186 TCHSV max – 8207 TKVCH min ⩾ 0

125 ns – 55 – 20 = 50 ns

PE is a chip select for a valid address range. It can be generated from the address bus or from the 80186's programmable memory selects. This design uses an inverted A19. The timing is determined by the interval between the address becoming valid and the falling clock edge, minus setup and delays.

80186 address valid to 8207 falling clock edge – 80186 address from clock delay – 8283 latch delays – 8207 PE setup ⩾ 0

1 TCLCL – 80186 TCLAV max – 8283 IVOV @ 300 pf – 8207 TPEVCL ⩾ 0

125 ns – 44 – 22 – 30 = 29 ns

The hold times are 0 ns and are met.

Address Setup

For an 80186 design, the 8207 requires the address to be stable before RAS goes active, and to remain stable for 2 clocks. Unused 8207 address inputs should be tied to Vcc.

tASR is a RAM specification. If it is greater than zero, this must be added to the address setup time of the 8207. Address setup is the interval between addresses being issued and RAS going active, minus appropriate delays.

80186 address valid to 8207 RAS active – 80186 address from clock delay – bus delays – (8207 setup + RAM t_{ASR}) ⩾ 0

TCLCL + 8207 TCLRSL min @ 150 pf(1) – 80186 TCLAV max – 8283 IVOV max @ 300 pf – (8207 TAVCL min + DRAM t_{ASR}) ⩾ 0

125 ns + 0 – 44 – 22 – (35 + 0) = 24 ns

The address hold time of 2 clocks + 0 ns is always met, since the addresses are latched by the 8282/3. Even when the processor is in wait states (for refresh),

(1) Not specified—use 0 ns.

TCLCL + TCLCH + 8207 TCLW min(1) + 74S37 delay tPHL min @ 50 pf + additional loading (142 pf) - 80186 TCVCTV - 74S240 tPZL - bus delays (250 pf) - 74S240 delay - 2164A tDS ≥ 0

125 + 62.5 + 0 + 6.5 + 3.5 - 70 - 15 - 7 - 7 - 0 = 98.5 ns

The hold time, tDH, is from WE going low to the 80186 DEN going high plus buffer delays minus WE from clock delays.

TCLCL - 80186 TCVCTX min + 74S32 tPD(2) min + 74S240 tPHZ (min)(2) + 250 pf bus delays + 74S240 propagation delay min - 8207 TCLW max - 74S37 tPHL @ 50 pf - 142 pf loading delays - DRAM tDH ≥ 0

62.5 ns + 10 + 2 + 3 + 7 + 3.5 - 35 - 3.5 - 30 = 19.5 ns

All margins are actually better by about 10-20 ns. No improvement in timing was allowed for lower capacitive loads when additional buffers are used (i.e. the 80186 address out delay is at 200 pf, but the 8283 latch only loads these lines with about 20 pf).

SUMMARY

The 8207 supports the 80186 microprocessor running with no wait states. The 8207 interfaces easily between the microprocessor and dynamic RAM. There are no difficult timings to be resolved by the designer using external logic.

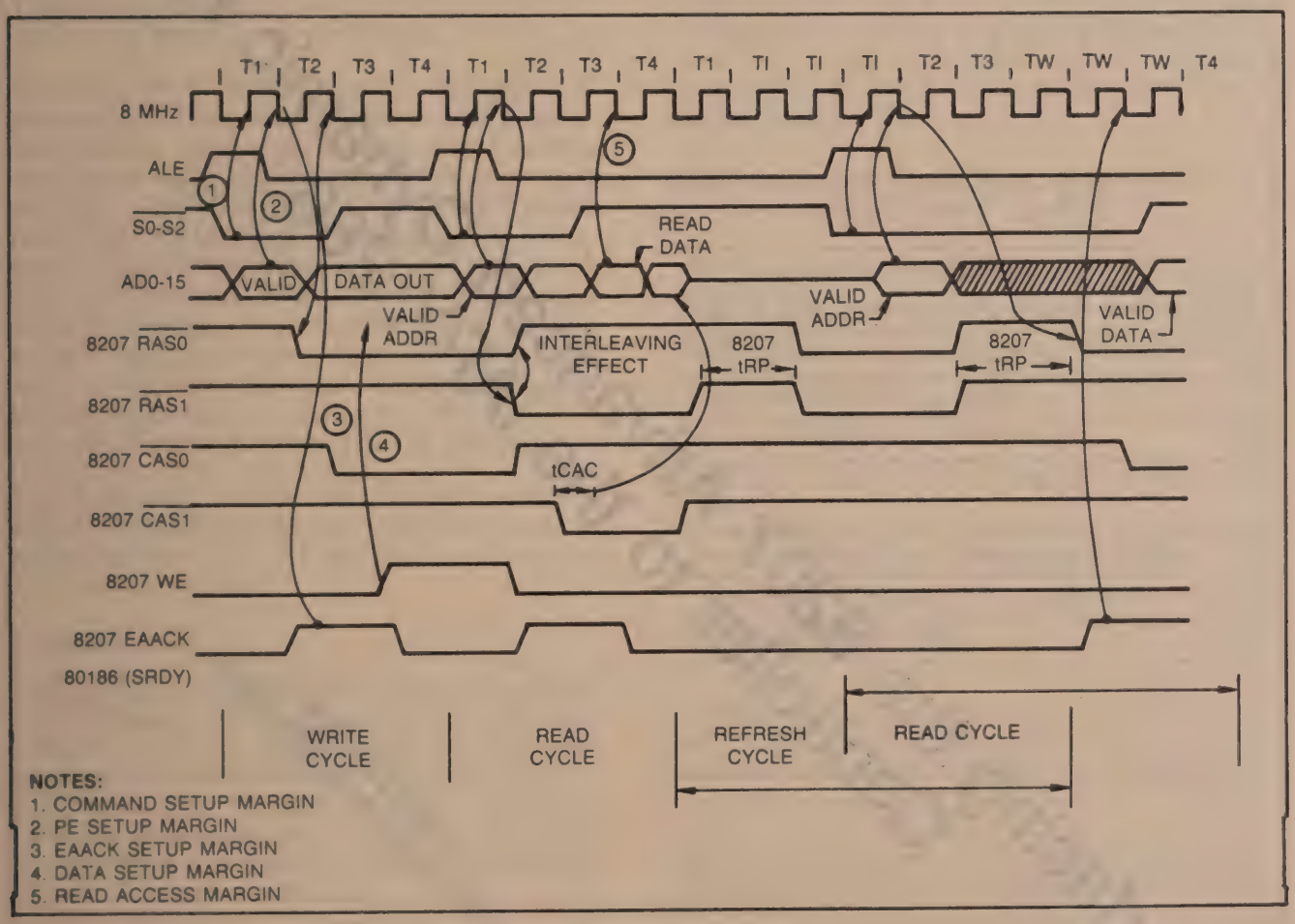


Figure 4. 8207/80186 timing relationship

(1) Not specified, use 0 ns.
(2) Not specified, use one half of typical value.

intel

APPLICATION NOTE

AP-168

August 1983

Interfacing the 8207 Advanced Dynamic RAM Controller to the iAPX 286

JIM SLEEZER
APPLICATION ENGINEER

©Intel Corporation, 1983.

ORDER NUMBER: 230862-001

NOVEMBER 1983

INTRODUCTION

The 80286 high speed microprocessor pushes microprocessor based systems to new performance levels. However, its high speed bus requires special design considerations to utilize that performance. Interfacing the 80286 to a dynamic RAM array require many timings to be analyzed, refresh cycle effects on bus timing examined, minimum and maximum signal widths noted, and the list continues.

The 8207 Advanced Dynamic RAM Controller was specifically designed to solve all interfacing issues for the 80286, provide complete control and timing for the DRAM array, plus achieve optimum system performance. This includes the normal RAM 8 warm-up cycles, various refresh cycles and frequencies, address multiplexing, and address strobe timings. The 8207 Dynamic RAM Controller's system interface and RAM timing and control are programmable to permit it to be used in most applications.

Integrating these functions (plus dual port and error correcting interfaces) allows the user to realize significant savings in both engineering design time, PC board space and product cost. For example, in comparing the 8207 to the ISBC012B 512k byte RAM board (where the DRAM timing and control is done entirely with TTL), the 8207 design saved board space (3 in^2 vs 10 in^2); used less power (420 ma vs 1220 ma); reduced the design time; and increased margins due to less skewing of timings. The comparison is based upon a single port 8207 design and does not include its RAM warm-up, dual port, error correcting, and error scrubbing or RAM interleaving features.

This Application Note will detail an 80286 and 8207 design. The reader should have read the 8207 and the 80286 data sheets, a DRAM data sheet*, and have them available for reference.

DESIGN GOALS

The main objective of this design is to run the RAM array without wait states, to maximize the 80286's performance, and to use as little board space as possible. The 80286 will interface synchronously to Port A of the 8207 and the 8207 will control 512k bytes of RAM (4 banks using 64k DRAMs). The dual port and error correcting features of the 8207 are covered in separate Application Notes.

8207 INTERFACE

The 8207 Memory design can be subdivided into three sections:

- Programming the 8207.
- The 80286/8207 interface.
- The Dynamic RAM array.

Programming the 8207

The RAM timing is configured via the 16 bit program word that the 8207 shifts-in when reset. This can require two 74LS165 shift registers to provide complete DRAM configurability. The 8207 defaults to the configuration shown in Table 1 when PDI is connected to ground. This design does not need the flexibility the shift registers would allow since standard 8207/80286 clock frequencies, DRAM speeds and refresh rates are used. Table 1 details the 8207/80286 configuration and Table 10 in the Data Sheet identifies "CO" as the configuration of the 8207 all timings will be referenced to (80286 mode at 16 MHz using fast RAMs = CO).

Table 1. Default Non-ECC programming, PD1 pin (57) tied to ground.

Port A is Synchronous (EAACKA and XACKA)
Port B is Asynchronous (LAACKB and XACKB)
Fast-cycle Processor Interface (10 or 16 MHz)
Fast RAM 100/120 ns RAM
Refresh Interval uses 236 clocks
128 Row refresh in 2 ms; 256 Row refresh in 4 ms
Fast Processor Clock Frequency (16 MHz)
"Most Recently Used" Priority Scheme
4 RAM banks occupied

The 8207 will accept 80286 status inputs when the PCTLA pin is sampled low at reset. This pin is not necessary for an 80286 design (besides programming) and is tied to ground.

Refresh is the final option to be programmed. If the Refresh pin is sampled high at reset, an internal timer

*All RAM references in this Application Note are based upon Intel's CMOS 51C64-12 64k Dynamic RAM. Any DRAM with similar timings will function. Refer to section 4.4.

is enabled, and if low at reset, this timer is disabled. The first method is the easiest to implement, so the RFRQ pin is tied to Vcc.

The differential reset circuit shown in the Data Sheet is necessary only to ensure that memory commands are not received by the 8207 when Port A is changed from synchronous to asynchronous (vice versa for Port B). This design keeps Port A synchronous so no differential reset circuit is needed.

RAM Array

The 8207 completely controls all RAM timings, warm-up cycles, and refresh cycles. To determine if a particular RAM will work with the 8207, calculate the margins provided by the 8207 (Table 15, 16—8207 Data Sheet) and ensure they are greater than the RAM requirement. An additional consideration is the access times of the RAMs. The access time of the system is dependent upon the number of data buffers between the 80286 and the DRAMs. To operate the 80286 at zero wait states requires access times of 100-120 ns. Slower RAMs can be used (150 ns) by either adding a wait state (programming the 8207 for "C1") or reducing the clock frequency (to 14.9 MHz approximately and maintaining the CO configuration.)

All write cycles are "late writes" and the data out lines of the RAM will go active. This will require separate data in and out lines in the RAM array. Another consideration for the RAM array is the proper layout of the RAM, and impedance matching resistors on the 8207 outputs. Proper layout is covered in Intel's RAM Data Sheets and Application Notes.

Microprocessor Array

To achieve no wait state operation, the 8207's clock input must be connected to the 80286's clock input. The EAACK (early acknowledge) output of the 8207 must connect to the SRDY input of the 82284. The 8207's address inputs connect directly to the 80286 address outputs and the addresses are latched internally. This latch is strobed by an internal signal with the same timing as LEN (which is for dual port 80286 designs). Figure 2 shows the timing relationship between LEN and the 80286.

LEN will fall from high to low, which latches the bus address internally, when a valid command is received. LEN can go high in two clock cycles if the RAM cycle started (RAS going low) at the same time LEN went low. If the 8207 is doing a refresh cycle, the 80286 will be put into wait states until the memory cycle can start. LEN will then go high two clocks after RAS starts, since addresses are no longer needed for the current RAM cycle. Thus the low period of LEN could be much longer than listed in the Data Sheet.

DESIGNING THE HARDWARE

Figure 1 shows a detailed block diagram of the design and Figure 2 shows the timing relationship between the 8207 and the 80286.

The following analysis of six parameters will confirm that the design will work. These six system parameters are generally considered the most important in any microprocessor—Dynamic RAM design.

8207 Command Setup Margin

Two events must occur for the 8207 to start a memory cycle. Either RD or WR active (low) and PE must be low when the 8207 samples these pins on a falling clock edge. If PE is not valid at the same clock edge that samples RD or WR active, the memory cycle will be aborted and no acknowledge will be issed.

The command setup time is based upon the status being valid at the first falling clock edge.

80286 status valid to 8207 falling clock − 80286 status from clock delay − 8207 command setup to clock ≤ 0

TCLCL − 80286 t12 (max) − 8207 TKVCL (min) ≤ 0

62.5 − 40ns − 20ns = 2.5ns

PE is decoded from the address bus and must be set up to the same falling clock edge that recognizes the RD, WR inputs. This margin is determined from the clock edge that issues the address and the clock edge that will recognize RD or WR, minus decoding logic delays.

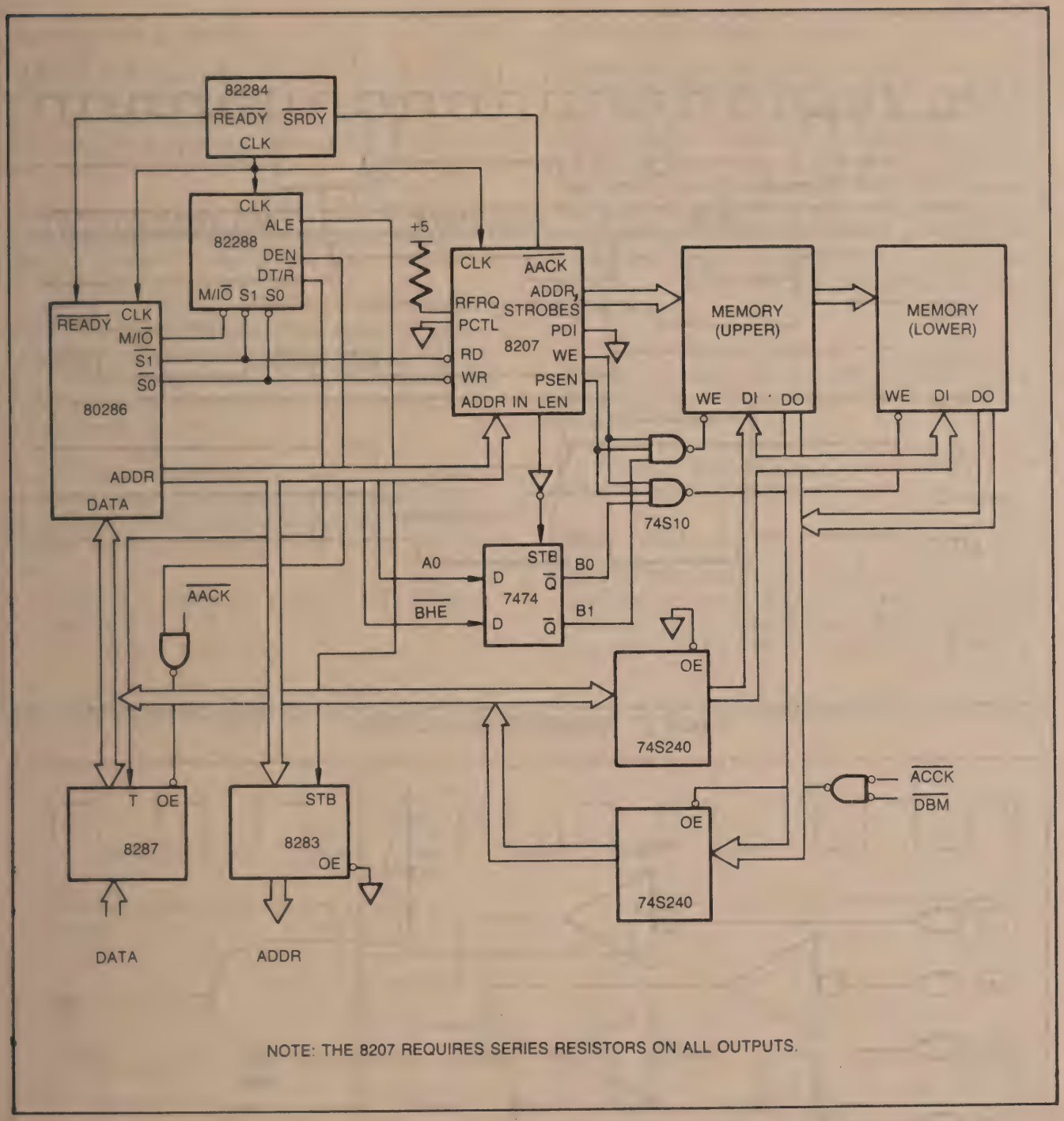
There are 2 clocks between addresses being issued by the 80286 and PE being sampled by the 8207. Then the 80286 address delay from the clock edge and decoding logic delays are subtracted from this interval. This margin must be greater than 0.

2TCLCL − 80286 t13 (max) − 8207 TPEVCL (min) ≤ 0

125 − 60 − 30 = 35ns

The address decode logic must use no more than 35 ns (and less is better). Figure 3 shows an easy implementation which uses a maximum of 12 ns.

The 8207 requires a zero ns hold time and is always met.



NOTE: THE 8207 REQUIRES SERIES RESISTORS ON ALL OUTPUTS.

Figure 1. 80286 to 8207, non-ECC, Synchronous System Single Port

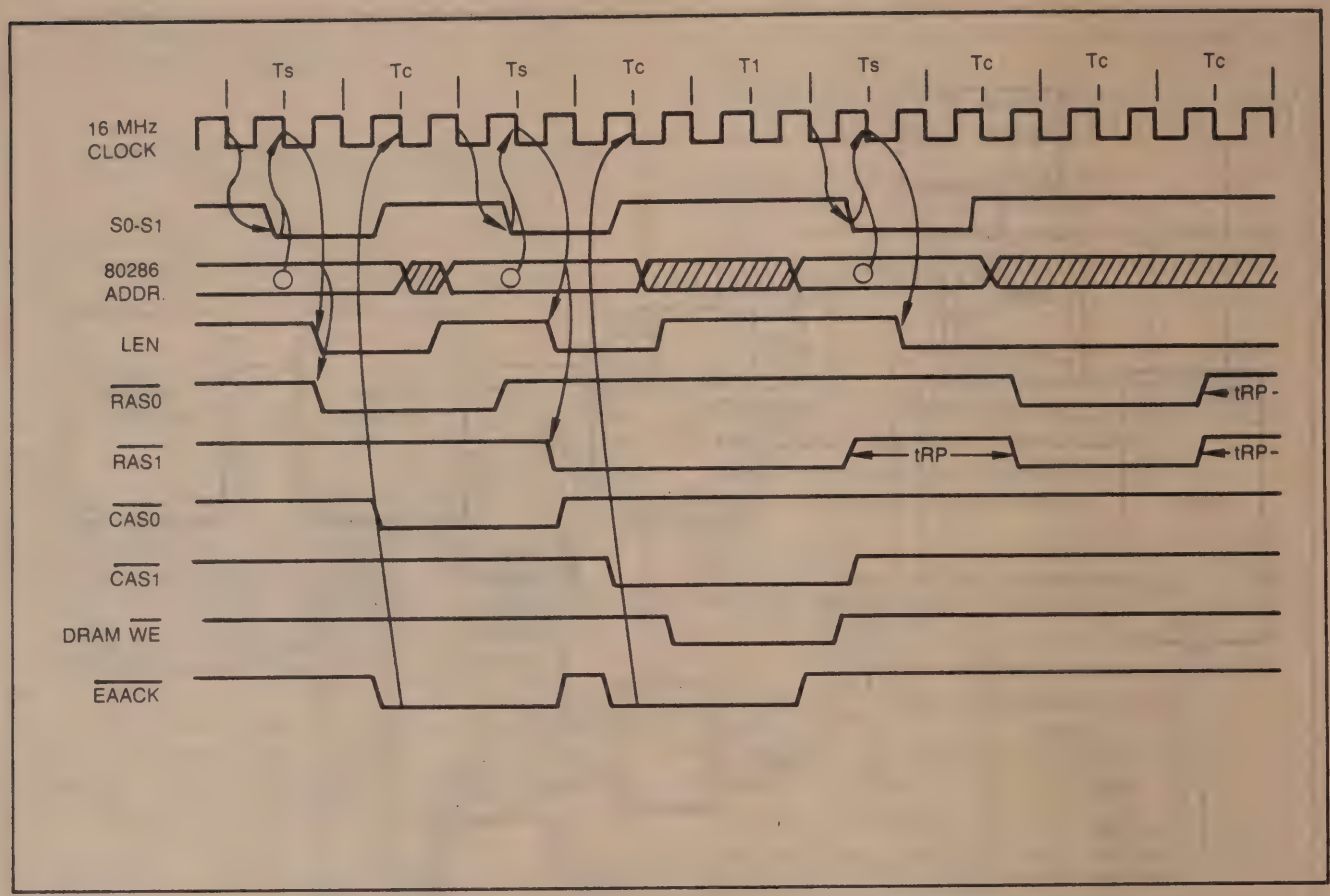


Figure 2. 80286/8207 Timing—"CO".

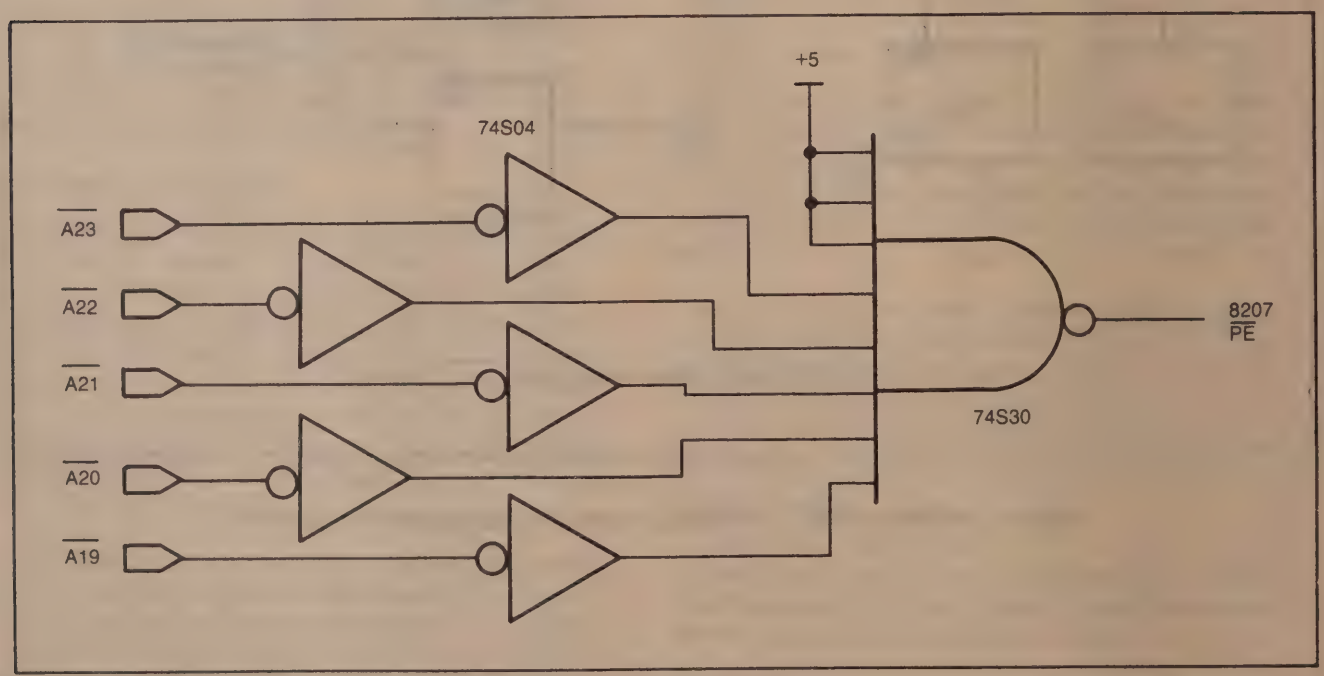


Figure 3. Address Decode Logic

Address Setup Margin

The 8207 must have stable addresses up to two clocks after RAS goes active. This is of no concern to the user, since LEN latches the address internally and will not admit a new address until two clocks after RAS goes active.

Addresses must be stable at least 35 ns (tAVCL) before RAS goes active to allow for propagation delays through the 8207, if a RAM cycle is not delayed by the 8207.

tASR is a RAM specification. If it is greater than zero, tASR must be added to the address setup time of the 8207. Address setup is the interval between addresses being issued, by the 80286, and RAS going active, minus appropriate delays.

The margin is determined from the number of clocks between addresses being issued from the 80286 to RAS going active. Exactly when RAS goes active is unimportant, since here we are interested only in the clock edge.

2TCLCL − 80286 t13 (max) − 8207 TAVCL (min) ⩽ 0
125 − 60ns − 35ns = 35ns

Acknowledge Setup Margin

The 8207 acknowledge (EAACK) can be issued at any point in the 80286 bus cycle (end of ϕ1 or ϕ2 of Ts or Tc). If EAACK is issued at the end of ϕ2 (Ts or Tc), the 80286 will complete the current bus cycle. If EAACK is issued at the end of ϕ1 of Tc, the 82284 will not generate READY to the 80286 in time to end the current bus cycle. A new Tc would then be generated and EAACK would now be sampled in time to terminate the bus cycle. EAACK is 3 clocks long in order to meet setup and hold times for either condition.

We need the margin between the 8207 issuing EAACK and the 82284 needing it. Figure 4, shows a worst case example.

TCLCL − 8207 TCLAKL max − 82284 t11 ⩽ 0
62.5 − 35 − 15 = 12.5ns

Read Access Margin

The 8207 will typically start a memory cycle (i.e. RAS goes low) at the end of ϕ1 of Ts. But if the start of a memory cycle is delayed (by a refresh cycle for instance), then RAS will be delayed. In the first case,

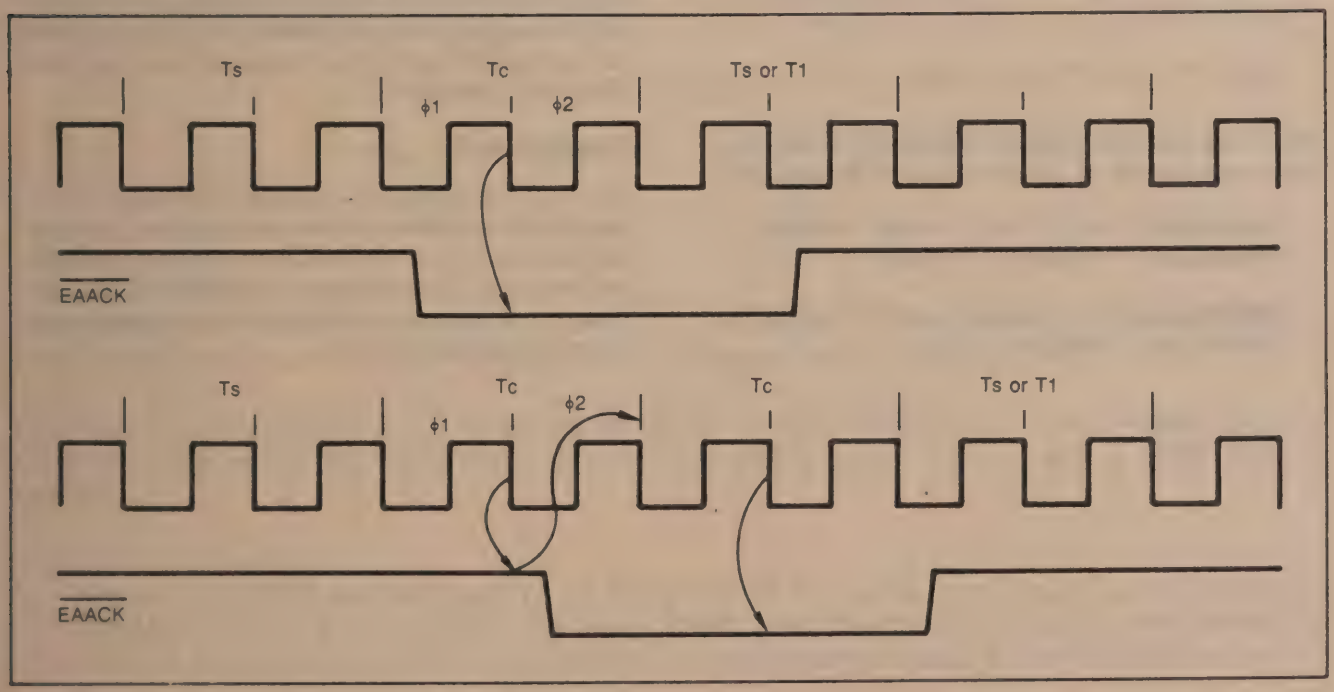


Figure 4. Acknowledge to the 82284

this represents 3 clocks and the second case could require 4 clocks to meet the data setup requirements of the 80286. In either case, data must be valid at the end of Tc. The 8207 holds CAS active long enough to ensure valid data is received by the 80286 in either case.

DRAMs specify two access times, RAS access (tRAC) and CAS access (tCAC) Both access periods must be calculated and the one with the least margin used. Also the number of data buffers should be kept to a minimum. Too many buffers would require either faster (more expensive) DRAMs, or a reduction in the performance of the CPU (by adding wait states).

RAS Access Margin

3TCLCL − 8207 TCLRSL max @ 150 pf − DRAM tRAC − 74S240 propagation delay max @ 50 pf − 80286 t8 ⩽ 0

187.5 − 35 − 120 − 7 − 10 = 15.5ns

CAS Access Margin

2TCLCL − 8207 TCLCSL max @ 150 pf − DRAM tCAA (or tCAC − 74S240 tplh max @ 50 pf − 80286 t8 ⩽ 0

125 − 35 − 60 − 7 − 10 = 13ns

By solving each equation for tRAC and tCAC, the speed requirement of the RAM can be determined.

DRAM tRAC = 3 TCLCL − 8207 TCLRSL − 74S240 tplh − 80286 t8 = 135.5ns

DRAM tCAC = 2 TCLCL − 8207 TCLCSL − 74S240 tplh − 80286 t8 = 73ns

So any DRAM that has a RAS access period less than 135 ns, a CAS access period less than 73 ns, and meets all requirements in the DRAM Interface Timing (Table 15, 16—8207 Data Sheet), will work.

Write Data Setup and Hold Margin

Write data from the processor must be valid when the 8207 issues WE to meet the DRAM specification tDS and then held to meet the tDH requirement. Some write cycles will be byte writes and the information to determine which byte is decoded from A0 and BHE/. Since the 80286's address bus is pipelined, these two signals can change before the RAM cycle starts, hence they must be latched by LEN. PSEN is used in the WE term to shorten the WE pulse. Its use is not essential.

Data must be set up to the falling edge of WE, since WE occurs after CAS. The 2 clocks between valid write data and WE going active (at the RAM's) minus propagation delays determines the margin.

2 TCLCL − 80286 t14 (max) @ 100 pf − 74S240 tplh + 8207 TCLW (min)[1] + 74S10 tphl @ 192 pf[2] − DRAM tDS = 0

125 − 50 − 7 + 0 + 14 − 0 = 82ns

The timing of the 8207's acknowledge is such that data will be kept valid by the 80286, for more than two clocks after WE goes active. This easily meets all RAM tDH specifications.

SUMMARY

The 8207 complements the 80286's performance and high integration with its own performance, integration and ease of use. No critical timings or logic design has been left to the designer. The 80286/8207 combination allows users to realize maximum performance from their simpler design.

1. Not specified. Assume no delay for worst case analysis.
2. STTL derated by .05ns/pf.

7231B

APPLICATION BRIEF

INTERFACING THE DYNAMIC RAM CONTROLLER TO THE iAPX 186

Jim Sleezer

1.0 INTRODUCTION

The 80186 microprocessor has integrated about 20 typically used system components into the same package as the microprocessor. This integration saves board space and design-in time. The 8208 Dynamic RAM Controller continues this system level integration. It is designed to control up to 256 Kbytes of Dynamic RAM (DRAM) using 64 K x 1 DRAMs, and up to 1 Mbyte using 256 K x 1 DRAMs.

Besides generating all DRAM control and timings, the 8208 allows various refresh types, frequencies, and microprocessor interfaces. Additionally, the 8208 does the 8 DRAM warm-up cycles back-to-back to prepare for operation.

By integrating the entire RAM timing and programmable refresh types, refresh rates, and interfaces into a single package, the user realizes significant savings in development time and board space. For example, a quick comparison of the 8208 versus a TTL implementation (using just the DRAM timing logic from Intel's iSBC012B memory board) yielded the following results:

1) a reduction in board space (10 in^2 to 3 in^2),

2) a reduction in power (1.2 A to 300 mA), and

3) much less design time (1 day).

The difference would be greater still if RAM warm-up, refresh, and interface programmability were added to the TTL implementation.

This Application Note will examine an 8208 to 80186 design. The reader should already have read the 8208 Data Sheet, the 80186 Data Sheet, and a DRAM Data Sheet*.

---

* While all DRAM references in this Application Note are based upon Intel's 2164A-15 64 K x 1 Dynamic RAM, any DRAM that meets the timing requirements in the Data Sheet, Table 8, and A.C. Characteristics, plus satisfies the Read Data Access Margin, will work.

2.0 HARDWARE DESIGN

An 8208 design can be divided into three areas: programming the 8208, DRAM compatibility, and system interface. While each topic will be covered in this Application Note, the 8208's programming logic defaults to an 8 MHz 80186 synchronous status interface with 150 ns access RAMs. All programming, RAM timings, and interface issues are satisfied for that configuration.

2.1.0 8208 PROGRAMMING

On the trailing edge of Reset, the 8208 samples the levels on two input pins and clocks in a 9 bit serial programming word. One input pin controls the type of refresh to be performed, while the other input pin alters the edge on which the 8208 samples memory commands. The program word further configures the 8208 for a refresh rate as a function of 8208 clock frequency, synchronous or asynchronous operation, and either an advanced acknowledge or Multibus compatible acknowledge.

2.1.1 REFRESH TYPES

If the REFRQ pin is sampled high at reset, an internal refresh timer is enabled; a low disables it. Both modes allow an external refresh cycle request by pulsing the REFRQ pin. An external request is generated by a low-to-high transition, and sampled by an 8208 (clock edge). Burst refresh occurs only when the timer is disabled and the REFRQ pin is sampled by two falling clock edges. The easiest method is to tie the REFRQ pin to Vcc (through a pull-up resistor); refresh cycles are transparent to the user.

2.1.2 8208 COMMANDS

The 8208 alters the point at which it samples a command and its response to the command inputs, based on the level sampled on PCTL when reset goes inactive. A high enables the status interface and a rising clock edge is used (this would be the middle of the T1 state; refer to the Timing Diagram). If low, the Multibus compatible interface is selected and a falling edge is used to allow for more propagation delay.

When the status interface is used, the status lines must be externally pulled up. The 80186 will tristate them when reset and the proper level (high) may not be seen by the 8208.

2.1.3 PROGRAM WORD

The program word defaults to a synchronous interface, fast acknowledge (for no wait states), and a refresh rate compatible with an 8 MHz clock (128 row/2 ms; 256 row/4 ms). When operating the 8208 at 8 MHz, most designs will not need to alter any programming bits and the PDI input pin can be tied to ground. If the 8208 is not run at 8 MHz a 74LS165-type shift register is needed to adjust for a proper refresh rate; otherwise, refresh cycles would not be performed often enough and data would be corrupted.

2.1.3.1 REFRESH RATE OPTIONS ($\overline{CIO}$, $\overline{CIT}$, $\overline{PLS}$, $\overline{FFS}$)

These four programming bits permit almost any DRAM to be used without wasting memory bandwidth. The combination of these four bits selects one of sixteEn clock intervals as shown in Table 1.

			Count Interval CI1, CI0 (8208 Clock Periods)			
CFS	PLS	FFS	00 (0%)	01 (10%)	10 (20%)	11 (30%)
0	1	1	118	106	94	82
0	0	1	59	53	47	41
0	1	0	74	66	58	50
0	0	0	37	33	29	25

Table 1. Refresh Count Intervals

3.0 HARDWARE DESIGN EXAMPLE

The objective is to have the 80186 run without wait states when accessing a DRAM array. The total amount of DRAM is 128K bytes and will be organized as 1 bank of 64K words.

Figure 1 is a block diagram of our design showing all relevant logic. The programming shift register is not needed since the 8208 will be operating at 8 MHz, and the other defult values are required. A data buffer is required in a no wait state design, since during reads the 8208 CAS line drives data onto the bus up to 50 ns past the end of T4. If another bus cycle were starting, then the multiplexed address/data lines would conflict with the driven data bus. This would reduce the systems' address to ALE setup margins. Figure 2 is a timing diagram of the design.

The timing parameters that are examined ensure that this portion of the system will operate properly. The parameters are:

1. Command setup and hold margin.
2. Address setup and hold margin.
3. Acknowledge setup and hold margin.
4. Write data setup and hold margin.
5. Read access margin.

3.1 ACKNOWLEDGE SETUP AND HOLD MARGIN

The 8208 early acknowledge (AACK) is intended to be connected to the SRDY input on the 80186 after being inverted. The AACK is issued at the beginning of T2 and must be valid at the beginning of T3.

1TCLCL - 8208 TCLAKL max - 7410 tPLH @ 15 pf - 80186 TSRYCL min _ 0

125 ns - 35 - 22 - 35
= 33 ns

The 80186 hold requirements, TCLSRY, of 15 ns is always met. The 15 ns hold time applies only when READY is being looked at by the 80186. Transitions that occur anywhere else in the bus cycle have no effect. AACK is two clocks long and is issued from a falling clock edge. AACK would always be sampled one clock into its duration. There would be a hold time of about 1 clock.

3.2 COMMAND SETUP AND HOLD MARGIN

Two events must occur for the 8208 to recognize a valid memory command. The 80186 status outputs are sampled by a rising clock edge (middle of T1 typically) and PE is sampled on the very next falling clock edge. If PE is not sampled at this point, no memory cycle will start. The status lines would have to go inactive before requesting another memory cycle.

The status setup margin is referenced to the middle of T4 or TI, and is required to be valid by the middle of T1.

1TCHCH - 80186 TCHSV max - 8208 TKVCH min _ 0

125 ns - 55 ns - 20 ns
= 50 ns

PE setup margin is referenced to the beginning of T1 and must be valid by the end of T1. PE selects the 8208 for a valid address range. It can be generated from either the address bus or using the 80186's programmable chip selects.

1 TCLCL - 80186 TCLCSV max - 8208 TPEVCL min _ 0

125 - 66 - 30
= 29 ns

Both PE and the RD, WR, and PCTL inputs require a 0 ns hold time to their respective clock edges.

The 8208 latches this information internally for cases when a refresh cycle delays a memory cycle from starting. Thus, a cycle will start when the refresh cycle finishes, even if the status signals have gone inactive. The hold margin is always met.

3.3 ADDRESS SETUP AND HOLD MARGINS

The 8208 requires the addresses to be stable before RAS goes active, and to remain stable for two clock periods thereafter. Unused address inputs should be pulled up to Vcc with a resistor.

The 8208 generates a margin of 0 ns minimum for the DRAM specification tASR when the 8208 specification TAVCL is met. If some DRAM is found that needs a more positive margin for tASR, then this requirement must be added to TAVCL.

The setup margin is between the clock edge that addresses are issued from to the 8208 issuing RAS, minus delays.

1 TCLCL + 8208 TCLRSL min[1] (@ 150 pf) - 80186 TCLAV max - 8282 IVOV max (@ 300 pf) - [8208 TAVCL min + DRAM tASR] _ 0

125 ns + 0 - 44 - 30 - (35 + 0)
= 16 ns

The 8208's address bus is divided into two halves. AL0-8 becomes the DRAM row address outputs and AH0-8 becomes the column addresses (64K DRAMs would need AL0-7 and AH0-7 connected to the address bus, AL8, AH8 would be tied to Vcc). Internally, the 8208 latches AH0-8 with CAS to provide for tCAH - column address hold time. This latching occurs near the end of T2 for read cycles and near the end of T3 for write cycles. When the RAM cycle is delayed due to refresh, the timing of AACK will ensure the two clock hold requirement. No equation is provided since this happens internally.

---

[1] Since this is not specified, 0 will be used for analysis only. Based upon design information this value would be about 20 ns.

3.4 WRITE DATA SETUP AND HOLD MARGIN

During write cycles, data from the 80186 must be valid at the DRAM when CAS goes low, and satify the DRAM tDS specification. Data must then be held valid and referenced to CAS long enough to meet the DRAM specification tDH. In this design example DEN is the limiting factor in the data setup margin. DEN is active before data is issued by the microprocessor, but there is a significant delay before the buffer is active. The result is that write data will be valid at the buffer before it is fully capable of transmitting data. The margin is referenced to the clock edge that issues DEN and the clock edge that issues CAS, minus delays.

TCHCL + 1 TCLCL = 8208 TCLCSL min (@ 150 pf) -
80186 TCVCTV max - 74LS245 TPZH max - DRAM TDS _ 0

55 + 125 + 62.5 - 70 - 40 - 0
= 132 ns

The hold margin is referenced to the edge that issues CAS and when valid data disappears. DEN is the controlling signal because it can go inactive before the data bus is floated by the microprocessor.

1 TCLCL + 1 TCLCH + 80186 TCVCTX min + 74LS245 TPLZ min[1] -
8208 TCLCSL max (@ 150 pf) - DRAM TDH _ 0

125 ns + 55 + 10 + 7.5 - 121 - 30
= 46.5 ns

The WE pulse length may cause problems with back-to-back bus cycles. Shortening the pulse width will not cause any other problems. The easiest solution is to factor in a shorter width signal, such as AACK, as is done in the design example.

---

[1] This parameter is not specified. For analysis, either assume 0 ns or use a more realistic value, such as one-half of typical.

3.5 READ DATA ACCESS MARGIN

The design example requires a buffer in the data path because the 8208 will not stop driving data onto the bus until after the end of T4. With back-to-back bus cycles this would cause bus contention and reduce address to ALE setup margins. The DRAM access parameter used is called "TCAC", and is referenced from the CAS active edge - not RAS. This parameter varies widely between manufacturers. When analyzing read access margins, some trade-off between buffer speed and TCAC delays must be considered.

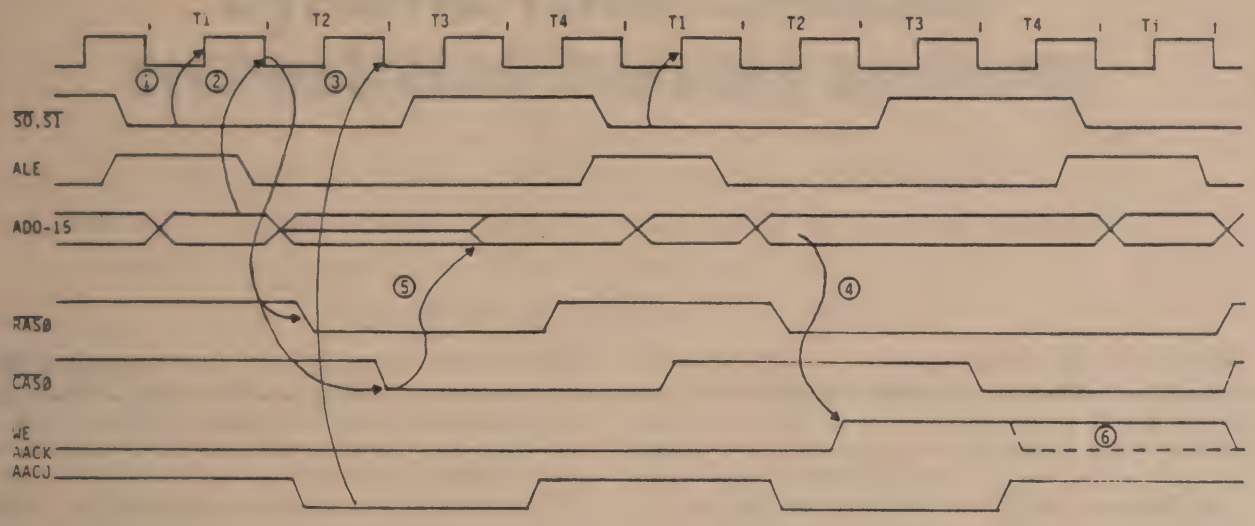
The 8208 starts a memory cycle, typically, at the end of T1, and data must be valid at the end of T3. With [refresh cycle] delayed bus cycles, data would still have to be valid in two clocks. The timing of the AACK signal guarantees this. From this two clock margin, buffer delays, TCAC delays, and others must be subtracted.

2 TCLCL - 8208 TCLCSL max (@ 150 pf) - DRAM TCAC max (@100 pf) - buffer delays max - 80186 TDVCL min _ 0

250 ns - 121 - 85 - 12 - 20
= 12 ns

4.0 SUMMARY

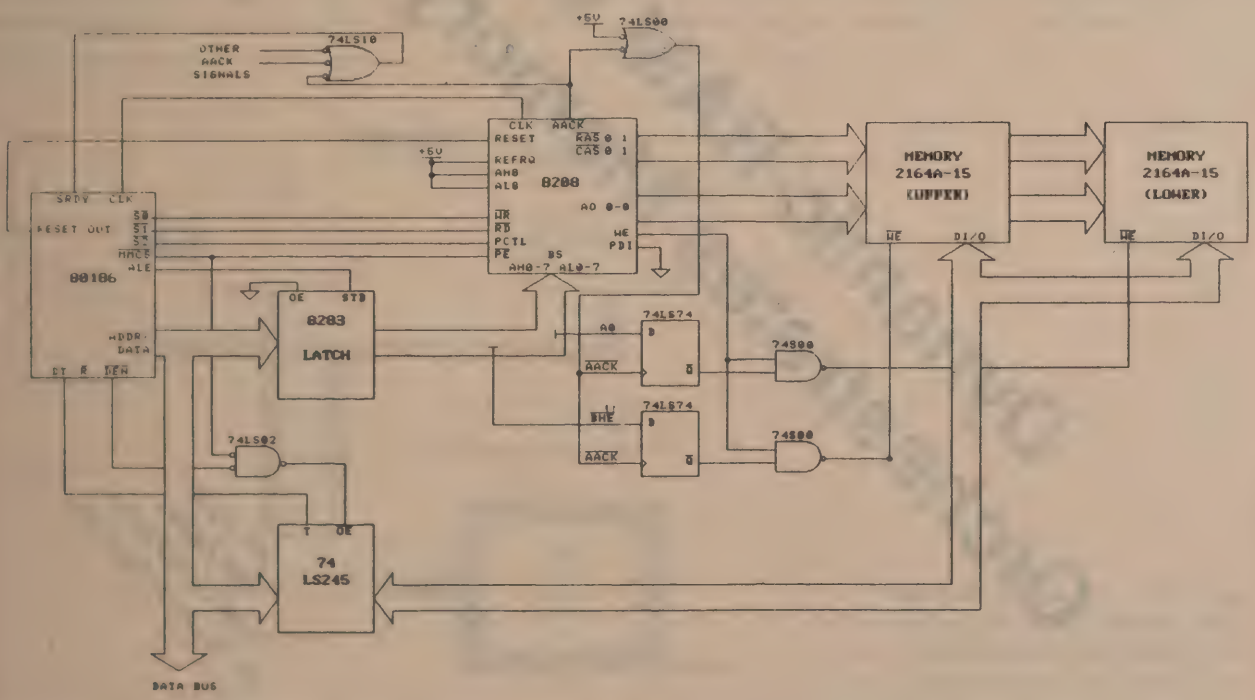
The 8208 solves most of the many design issues faced when adding a dynamic RAM array by giving the designer options. Options for various types of DRAMs, clock speeds, and system configurations. The margins that were just examined showed that the 8208 has plenty of margin in a system. Several margins were even higher. The READ DATA ACCESS MARGIN, for example, is considerably greater. The access time for DRAMS is specified with 100 pf loads, yet this was not added into the equation. Each designer should verify this analysis as specifications from manufacturer's change, without notice.



8208 Timing Diagram

Read then Write Cycle with one Bank of RAM

1. Status setup margin
2. Address setup margin
3. Acknowledge setup margin
4. Write data setup margin
5. Read access margin: based upon TCAC
6. Shortened WE pulse



intel

ARTICLE
REPRINT

AR-231

October 1982

Dynamic-RAM Controller Orchestrates Memory Systems

Jim Nadir
Mel Bazes
Intel Corporation
Santa Clara, California

"Reprinted from ELECTRONICS, Sept. 8, 1982. Copyright (c) McGraw-Hill, Inc. 1982. All rights reserved."

OCTOBER 1982
ORDER NUMBER: 210758-001

Dynamic-RAM controller orchestrates memory systems

Up to 88 chips take their cues from an n-channel MOS IC that both housekeeps and supports error-corrected dual-port memories

by Jim Nadir and Mel Bazes, *Intel Corp., Santa Clara, Calif.*

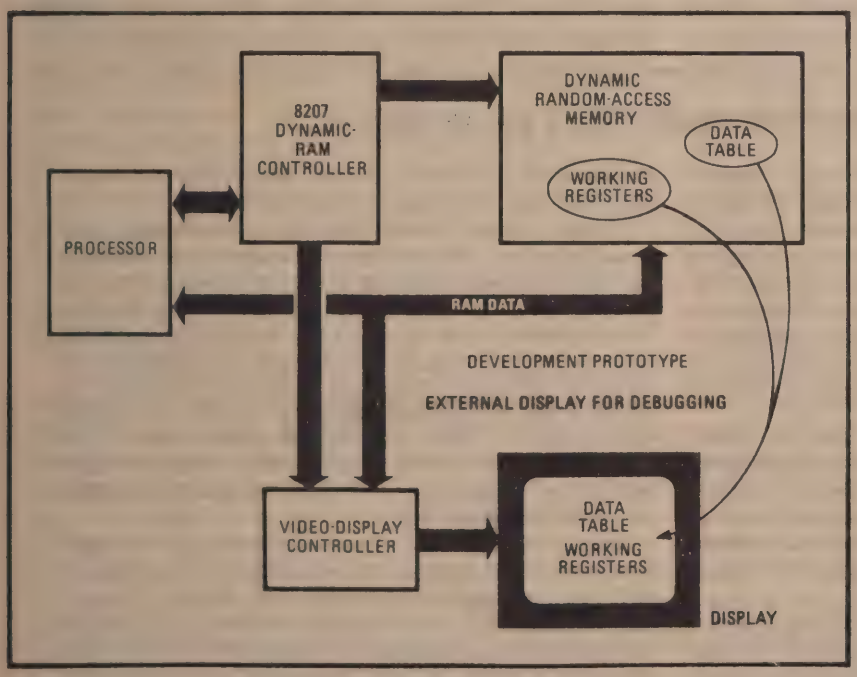
☐ Designing a dynamic–random-access-memory system means balancing the goals of high performance, reliability, and versatility against the often contrary aims of economy, simplicity, and compactness. In the last five or so years, the advent of dynamic-RAM controller chips relieved designers of some of the onus of tending to the needs of dynamic chips: standard supportive integrated circuits brought together the counters, timers, multiplexers, and other elements needed.

But controllers diverged into two types. One bought the high performance to ride with fast memory systems at the expense of functionality, while the other took on more and more functions to do a complete but slower job. The 8207—an advanced dynamic-RAM controller—blunts the horns of that dilemma and also solves a variety of less severe design problems.

A dynamic-RAM controller is charged with making a dynamic memory system appear static to the host processor. At a minimum, therefore, the controller takes over refreshing the memory chips, multiplexing the row and column addresses, generating control signals, timing the precharge period, and signaling the processor when data is available or no longer needed. But, beyond those local housekeeping chores, the controller can also go a long way to solving more global design problems, like sharing memory between two processors, not to mention detecting and correcting errors.

To realize this potential for a highly integrated solution, the 8207 has a dual-port interface and, when used with the 8206 error-checking and -correction unit, ensures data integrity in large dynamic-RAM systems. In addition to doing the jobs of refreshing, address multiplexing, and control timing, the unit supports memory-bank interleaving for pipelined accesses, overlaying RAM and read-only-memory locations, and initializing RAM.

The exact implementation of most of these functions is programmable, letting designers tailor their systems in detail. Systems containing up to 88 dynamic-RAM chips—whether 16-, 64-, or 256-K versions—in one, two, or four banks need only a single 8207 and no external buffering. Attesting to the high performance claimed, the 8207 mates dynamic RAMs having 100-nanosecond access times to the iAPX-286 processor operating at 8 megahertz without introducing any wait states.



1. Window on a micro. One use for a dual-port memory shared by independent processors is the development system shown. Adding a video display to the prototype itself gives a window on the system memory.

210758-001

To achieve that speed and include all those functions, the 8207 relies on a dense, high-speed n-channel MOS process (H-MOS II) and requires a chip some 230 by 200 mils in area. To meet the rigors of operation with even faster processors, novel logic and integrated-circuit designs are employed. Replacing the two-phase logic common in n-MOS ICs, single-phase edge-triggered logic simplifies logic and circuit design, precludes problems of clock-pulse overlap, and reduces the sensitivity to clock high and low times. Voltage-controlled capacitive loads form the delay elements that time critical output pulses, such as the address strobes, and compensate the output-switching delays for variations in power-supply voltage, temperature, and processing.

A low 20-ns setup time for input signals is achieved by cutting the RC delay of input-protection devices and moving the TTL-to-MOS signal buffering from the input pads to the pulse generators. A short 35-ns delay from input to output switching is achieved by triggering the output generators directly from the external clock, saving a buffer delay time. With the resulting high-speed performance and a high level of integration, the 8207 successfully attacks the stringent requirements of today's memory systems.

One system feature gaining popularity currently is the use of multiple processors operating on shared data to obtain higher performances and reliability. For example, a separate processor dedicated to input/output tasks frees the main processor for full-time data processing. Alternatively, multiple main processors can execute different tasks simultaneously. In all such cases, sharing a common memory space among the cooperating processors is the key to effective operation.

Unfortunately, when more than one processor accesses shared memory through a single bus, the limited bus bandwidth and the time spent in exchanging bus control slow down data transfers. Dual-port memory systems overcome this limitation by giving two processors access to a common memory through two independent buses. The 8207 includes a dual-port interface to simplify the design of shared memory systems.

Two-port memories can be used with multiprocessing or multitasking architectures. In the former, independent processors run independent programs, sharing only a common memory. Multitasking processors cooperate on different parts of the same task.

An example of a multiprocessing architecture is the dynamic video display (Fig. 1) that provides a window on a processor's memory. Centering the display over a data table, for example, immediately reveals how program execution affects the data, which aids in debugging programs. If a microcomputer is implemented with a dual-port memory—the second port for a dynamic video display—then the prototype itself can serve as a development and debugging system, reverting to single-port operation in the final version.

A dual-port architecture in a multitasking environment, on the other hand, adds a margin of safety to a shared-resource bus, such as Intel's Multibus. Although one of the biggest benefits of such a bus is the sharing of expensive peripherals among several users' programs, an intimidating problem is that a single program gone haywire can easily corrupt the entire system. A two-port memory, properly configured, circumvents this occurrence. Because each port has its own address, data, and control lines, problems on one side are confined by hardware to that side.

Port of call

As a general rule for multitasking architectures, one port of a two-port memory operates in a local environment, and the other port runs remotely, off the expandable shared-resource bus. The local processor is likely to require a synchronous port to reap the benefit of higher performance. Remote buses, in contrast, are usually configured asynchronously. Unless programmed other-

Dynamic-RAM controllers get in step

Synchronous and asynchronous signals have different requirements for interfacing with a controller. The terms synchronous and asynchronous are conventionally applied to dynamic random-access memory depending on whether it exists in a local or a remote environment, respectively. However, they more properly characterize the dynamic-RAM controllers, for the RAMs themselves need no clocks—the only restrictions as to the start of a memory access cycle involve ensuring that the refresh and precharge requirements are satisfied.

Because the controller decides both when to refresh and whether or not precharge and other timing requirements have been met, it does need a clock. Incoming commands can either always arrive with a fixed relationship to the controller's clock or have no particular relationship to it. The former are, of course, synchronous operations, the latter asynchronous.

The major difference between an asynchronous and a synchronous controller (or port of a controller, in the case of the dual-port 8207) is that the asynchronous controller must first synchronize the incoming commands to its own internal clock. From that point on, the asynchronous controller looks just like a synchronous device.

Whereas various techniques for synchronization are available off chip, on-chip synchronization is restricted to the resolution and sampling of states of a flip-flop. The incoming command is clocked into a resolving flip-flop. After a predetermined time, a sampling flip-flop reads the state of the resolving flip-flop, thereby synchronizing the command. Assuming that both flip-flops are triggered on the same edge of the controller's internal clock, the fastest that an asynchronous signal can be synchronized is one clock period. The slowest synchronization takes two clock periods; on the average, getting the signals in step takes one and a half clock cycles.

Because the processor typically requires four or fewer clock periods to complete a cycle, adding a cycle and a half for synchronizing increases the access time by approximately 25%. Synchronous controllers are therefore always preferred when the environment permits them, and local environments, such as single-board computers, generally do so.

wise, the 8207 configures one port synchronously, and the other asynchronously. For specific applications, both ports may be programmed as either synchronous or asynchronous (see "Dynamic-RAM controllers get in step," p. 129).

Whether the ports are programmed for synchronous or asynchronous operation, some mechanism must decide which processor will gain access to memory when both request it almost simultaneously. That mechanism consists of arbitration logic that controls access and always leaves one port selected. When a port is selected, its associated control and interface signals are passed directly to the RAM timing logic by the command multiplexer (Fig. 2). Both ports' command and control lines, after being synchronized, go into both the command multiplexer and the arbitration logic.

However, the arbitration logic enables the command multiplexer to pass only commands that appear at the selected port. At the same time as a command appears at a selected port, arbitration logic initiates the cycle-control logic that completes the timing of the RAM cycle that ensues. If a command appears on the unselected port, it will not get through the multiplexer to initiate a RAM cycle but will instead wait in the status-command decoder until the current command is completed, at which time the command multiplexer switches to the unselected port. The arbitration logic will then service this queued access request by starting a new cycle.

The arbitration logic examines all port requests, including the internal refresh port. The refresh-request port is subject to arbitration like the other two ports, except that it is always assigned a higher priority than an unselected external access port. Thus, refreshing can be delayed, at most, one RAM cycle.

While the current RAM cycle is running, the arbiter determines the next cycle to be initiated. Thus, the arbitration time of two or more simultaneous port requests is hidden by the memory cycle time. In other words, in cases where both a selected and an unselected port request access simultaneously, the arbitration time for the unselected port does not extend that port's access time, which is delayed by one memory cycle anyway. Only when an unselected port requests a free memory does the arbitration time slow access, because then the command must pass through the arbitration logic before a RAM cycle can be initiated. To minimize such delays in most cases, there are two arbitration algorithms to be selected by the user.

The first algorithm, intended for multiprocessing environments, automatically returns the arbiter to a designated preferred port, generally the higher-performance, synchronous port. Thus any command on the selected port generally has immediate access, whereas any command arriving at the unselected port must wait.

The second, or last-accessed-port, algorithm, which is applicable in multitasking environments, leaves the most recently accessed port as the selected port. This algorithm optimizes port selection for task passing in a multitasking environment. In task passing, the host processor sends a task to an execution processor; until the task is received, the execution processor seldom accesses memory. Conversely, once the task is passed, the host processor seldom accesses memory until the task is completed. Thus, the ports are used in spurts.

Because timely refreshing is needed to preserve dynamic-RAM data, a refresh request is always serviced on the next available cycle. The refresh algorithm, however, may be selected by the user. The options available are: no refresh, user-generated single refresh, automatic refresh, or user-generated burst refresh.

No refresh would be selected for applications like bit-mapped–video displays, where continuous, sequential access of all RAM locations itself refreshes every cell periodically. User-generated refresh modes allow the designer greater control over power dissipation, for example, in large memory systems. Automatic refreshing, in which the controller itself times the refresh interval and initiates the operation, lets the designer ignore the refresh requirements entirely. As mentioned, the refresh requests are subject to arbitration just like other access requests. However, once a burst refresh is selected, it remains active until completed.

Cleaning up errors

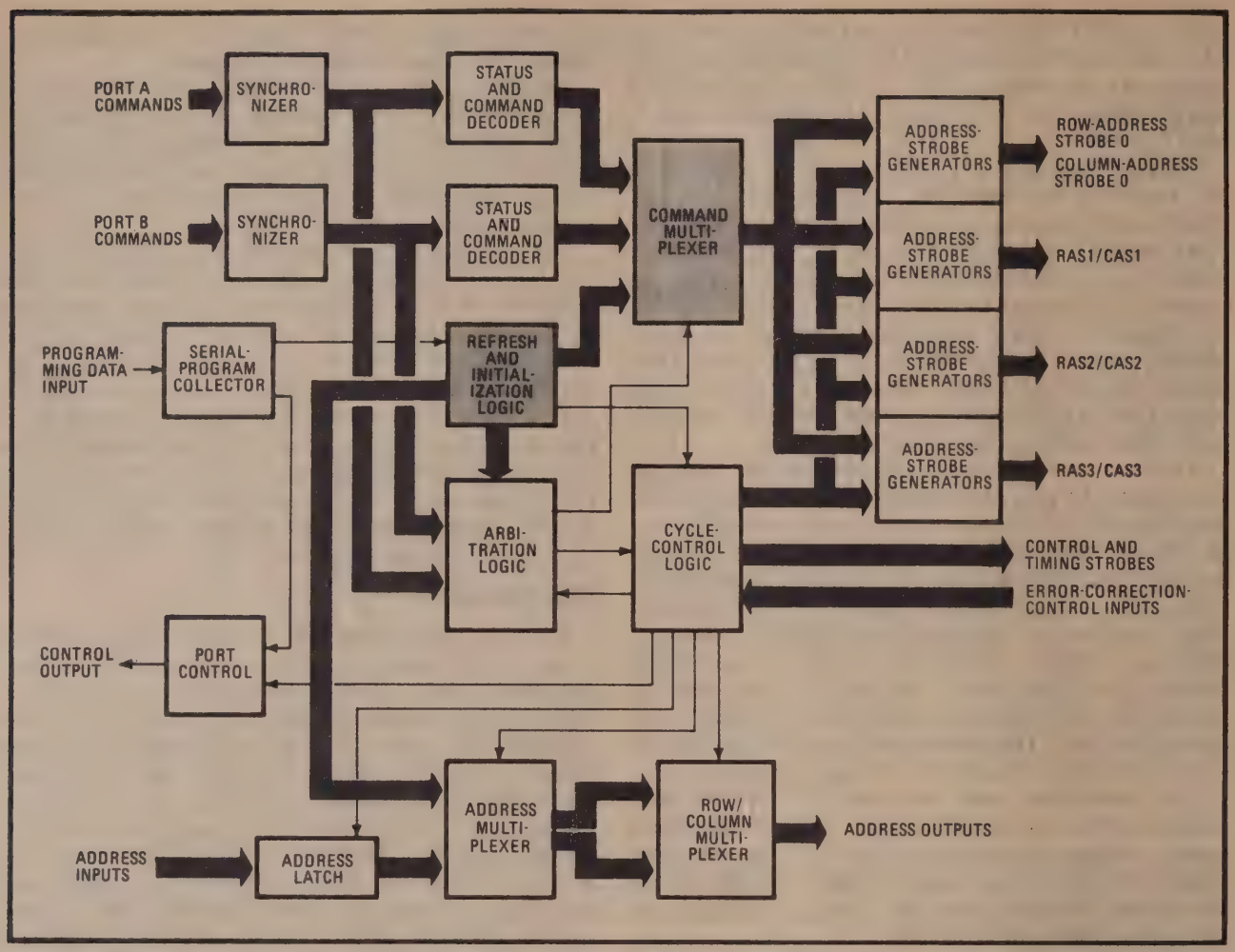
Ensuring data integrity is a major concern in large dynamic-RAM systems, particularly because of their susceptibility to soft errors caused by alpha-particle radiation. Various parity encoding techniques have been developed to detect and correct memory-word errors [*Electronics*, June 2, 1982, p. 153]. The parity bits, called check bits when used for correction as well as detection, are stored in the memory array along with their associated data word. When the data is read, the check bits are regenerated and compared with the stored check bits. If an error exists, whether in the retrieved check bits or in the retrieved data word, the result of the comparison—called the syndrome—gives the location in the group of the bit in error.

Two drawbacks surface in the design of any memory system that is to be protected by error-correction circuitry. First, the memory-word width must be increased to store the check bits; second, extra time must be allotted for the error-correction circuitry to generate the check bits on write cycles, plus more time to regenerate and compare the check bits on read cycles. The 8207 provides several ways to minimize both problems.

Error-correction schemes require a smaller proportion of check bits to protect wider memory words. For example, an 8-bit word needs 5 check bits, for a 63% increase in memory. Put the other way around, 38% of the available memory would be dedicated to the check bits. Six check bits are required to protect a 16-bit data word—only a 27% overhead. Clearly, the wider the memory array, the more economical the error correction.

The 38% overhead necessary to protect such 8-bit–bus machines as the 8088 or 8085 makes error correction an unattractive proposition. However, if the memory width could be doubled, with the 8088 accessing only half a word at a time, the overhead would drop to 27%.

Reading a double-width word, checking for soft errors, and then sending the desired portion of the word to the processor presents no major problems, unlike writing to such an array. The check bits cannot be calculated from only a portion of the word—they must be calculated for



2. Arbiter's labor. Two external ports plus the internal refresh port can request access to the memory system at once. Arbitration logic decides which to service, based on programmable algorithms. High-speed logic design cuts the delay from input to output switching to 55 ns.

the entire word at once. Whenever the processor writes a partial word to memory, it must first read the entire word, check it, substitute for that portion of the word to be rewritten, and recalculate the check bits. Only then can the entire word be written to memory. The 8207, working in conjunction with the 8206 error-checking and -correction unit, contains mechanisms to expedite this potentially arduous process.

Whenever the 8207 performs a partial-write cycle, it initiates a read-modify-write cycle wherein the entire memory word is first read and latched into the 8206 (Fig. 3). After the retrieved data has been verified as correct, new data is supplied to the RAM, half from the processor and half from the 8206, which also generates the check bits for the entire new word.

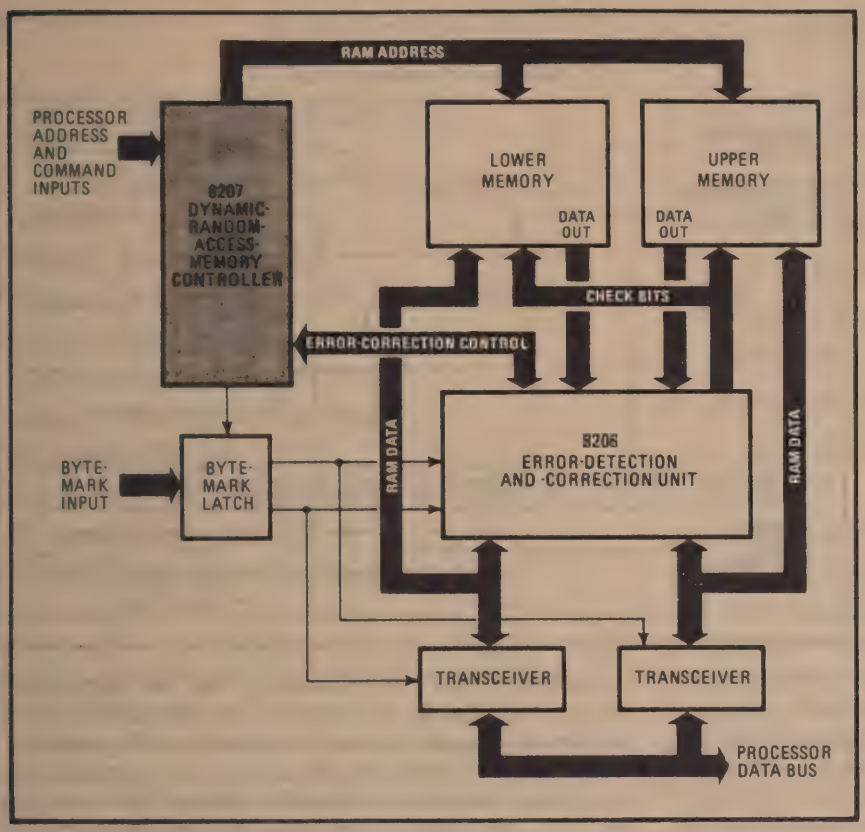
Control signals—called byte marks—specify which portion of the new data word is coming from the processor and which from the 8206. The byte marks determine whether the processor or the 8206 drives the RAM data bus—for example, if the 8206 is driving one portion of the data bus, the processor is prevented from driving the same portion. The byte-mark signals simply disable the appropriate transceivers. If, on the other hand, the processor is driving a portion of the RAM data bus, the byte marks change the 8206 data outputs to inputs, allowing the 8206 to read the data from the processor and calculate new check bits.

The ability of the 8207 to handle memories organized as one, two, or four banks allows tradeoffs between the cost and performance of an error-correction system. For maximum performance, memory would be organized in four banks, each 16 bits wide. In applications requiring error correction, but where maximum performance is not critical, concatenation of RAM banks into two banks of 32-bit words, or even one bank of 64-bit words, can make error correction very economical.

Holding to high performance

Even though the cost of error correction has thus been reduced to where it becomes an attractive solution, the problem remains of minimizing performance degradation. Tackling that challenge depends on the particulars of the configuration, such as whether the memory is to be used with a high-performance local processor, as system memory on a shared-resource bus, or is to be shared between a local high-performance processor and a shared-resource bus.

The method chosen to handle errors depends on the type of bus. Intel's Multibus is the kind that requires data to be valid prior to the issuance of a transfer-



3. Teamwork. The 8206 error-correction chip joins forces with the random-access-memory controller so that an 8-bit–bus processor may utilize the 16-bit-wide memory that is more economical for error-correction schemes. Byte marks configure the data buses for partial-word transfers.

acknowledge signal, in contrast to the local buses of the iAPX-86, -186, and -286 processors. A local bus will usually be synchronous, with a single processor or coprocessor group attached to it; the processor characteristics are known, as is the processor's response to a transfer-acknowledge signal.

With Multibus and other shared-resource buses, the processor types that will eventually be connected are not known in advance, and the buses themselves are generally asynchronous. Hence the time between the transfer-acknowledge signal and data becoming valid is not known. Therefore, the rule with such buses is to acknowledge a transfer only when data is valid. (On some asynchronous buses, the acknowledgment is issued earlier to compensate for synchronization delay at the receiving processor.)

Two basic configurations for checking and correcting errors derive from these system considerations and the fact that it takes longer to correct data than to detect an error. One is for buses that connect to processors and coprocessors receiving a transfer acknowledge prior to data becoming valid, and the other for buses that connect to processors receiving a transfer acknowledge after data is valid. Both configurations are supported by the 8206–8207 team.

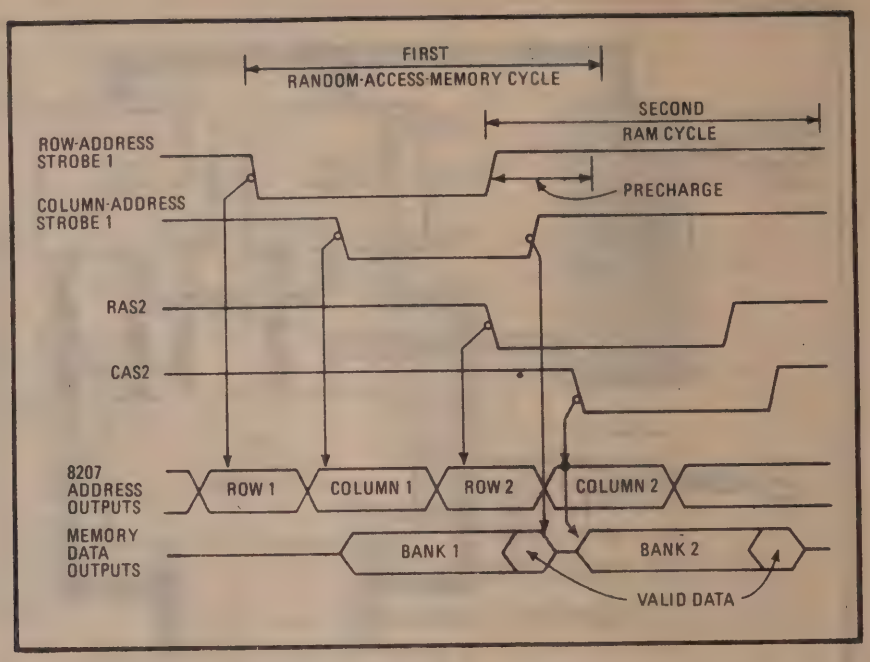
Buses among the former type of processors always get corrected data from the 8206, whether an error exists or not, and will carry a transfer acknowledge from the 8207 before data becomes valid on the bus. Though this means data is delayed for error correction on every transaction, the extra delay is immaterial, since it is hidden behind the processor's response time to the transfer-acknowledge signal. By the time the processor requires data, it is already corrected and on the bus. As a result, system performance is not degraded at all because of single-bit errors.

For buses among processors that receive the transfer acknowledge after the data is valid, the 8206 always checks for errors but does not routinely correct data. In this mode, RAM data passes through faster, because the 8207 will issue an acknowledgment sooner. If, however, an error is found, the 8207 will lengthen the cycle, command the 8206 to correct the data, and delay the transfer-acknowledge signal until the corrected data can be placed on the bus. For those buses with an acknowledge-synchronization delay, the 8207 can be programmed to issue the acknowledgment earlier to compensate for the delay.

Power-up problems

Another problem with memories protected by ECC circuits crops up when the power is turned on. At power-up, the data stored in memory is completely random; any attempt to read or perform a partial write will be aborted because the check bits will indicate multiple, and therefore uncorrectable, errors. For processors whose word width is the same as that of the memory array, the processor could simply initialize the entire memory array, taking some additional time and software. For memories whose word width is greater than that of the processor, however, initialization of the memory is not possible unless the error-checking or -correction circuitry is disabled by hardware, for example, by gating off the error flags.

The 8207 is equipped to deal with the initialization problem by itself. At system reset, the 8207 performs



4. Interleaving. Overlapping accesses to different banks increases memory throughput. Once the column-address hold time is satisfied, the 8207 starts a second cycle, pulling the second row-address strobe low.

eight cycles on all banks at once to warm up the dynamic RAMs, a typical RAM requirement for stable operation. The chip then individually initializes all memory locations to 0, adding the proper check bits. Though all memory banks could be initialized in parallel, that would require more power than any other memory operation, calling for a heftier and more expensive power supply needed only at system reset.

One final problem associated with memories protected by error-correction circuitry stems from the fact that only data that is accessed by the processor is corrected. If the processor continually accesses one particular segment of memory, the rest of the array may be accumulating soft errors. The possibility of two soft errors accumulating in a word of seldom accessed memory now becomes significant—and not all double-bit errors are correctable in simple ECC schemes. The 8207 scrubs memories to clean up this problem. During each refresh cycle, one word of memory is read, checked for errors, and if necessary, corrected before data is written back to memory. Because scrubbing occurs during refresh cycles with a read cycle replacing a row-address-strobe–only refresh cycle, no performance penalty is incurred. Scrubbing rids the entire memory of errors at least once every 16 seconds, reducing the probability of two soft errors accumulating in the same word almost to nil.

Bells and whistles

All dynamic RAMs require a recovery period for precharging internal lines after each access. If the processor were immediately to reaccess the RAM, the controller would have to delay it until the precharge time was over. By automatically organizing memory into banks so that sequential addresses are in different banks, the 8207 is usually able to hide the precharge time of one bank behind the access time of another. That organization follows from using the 2 least significant bits of the address to select the bank. Of course, a break in the program flow, such as would be caused by a jump or call instruction, raises the probability that the same bank may be immediately re-accessed. This probability is less in four-bank memories than in two-bank configurations.

Further performance advantages are gleaned by organizing memory into multiple banks. For example, the 8207 can speed throughput by pipelining cycles. Once the row and column addresses to one bank have been latched, the controller sends the row address for the next cycle to the next bank (Fig. 4).

The 8207's manifold features can be tailored to a given system with the use of a serial programming pin. This pin can either be strapped high or low to select one of two default modes or be programmed by means of a shift register. The external register is completely controlled by the 8207, eliminating any local processor support. Sixteen bits are shifted into the 8207 to configure up to nine different features. The bits are arranged in order of increasing importance; using a shift register with less than 16 bits permits just those features needed to be programmed.

Programmable features of the processor interface include the choice of arbitration algorithm, clock compensation, and preferred port. At the RAM interface, the user can specify fast or slow memory chips, indicate bank configuration, and select the optimal refreshing scheme. In anticipation of the next generation of 256-K dynamic RAMs, the 8207 can support a 256-row–1-millisecond refresh convention, in addition to the 128-row–2-ms one for current 16- and 64-K parts.

Helping facilitate system design is a self-programming processor interface. By decoding the command input pins at power-up, the 8207 automatically determines whether it is connected to the status lines of an 8086, iAPX-286 or to the command lines of the Multibus. Because the 8207 can directly decode the status lines of Intel microprocessors, it can anticipate the next memory cycle and start a new cycle before actually receiving a command. This extra pipelining enables the designer to specify slower RAMs then would otherwise be required.□

intel®

8231A
ARITHMETIC PROCESSING UNIT

- Fixed Point Single and Double Precision (16/32 Bit)
- Floating Point Single Precision (32 Bit)
- Binary Data Formats
- Add, Subtract, Multiply and Divide
- Trigonometric and Inverse Trigonometric Functions
- Square Roots, Logarithms, Exponentiation
- Float to Fixed and Fixed to Float Conversions
- Stack Oriented Operand Storage
- Compatible with all Intel and most other Microprocessor Families
- Direct Memory Access or Programmed I/O Data Transfers
- End of Execution Signal
- General Purpose 8-Bit Data Bus Interface
- Standard 24 Pin Package
- +12 Volt and +5 Volt Power Supplies
- Advanced N-Channel Silicon Gate HMOS Technology

The Intel® 8231A Arithmetic Processing Unit (APU) is a monolithic HMOS LSI device that provides high performance fixed and floating point arithmetic and floating point trigonometric operations. It may be used to enhance the mathematical capability of a wide variety of processor-oriented systems. Chebyshev polynomials are used in the implementation of the APU algorithms.

All transfers, including operand, result, status and command information, take place over an 8-bit bidirectional data bus. Operands are pushed onto an internal stack and commands are issued to perform operations on the data in the stack. Results are then available to be retrieved from the stack.

Transfers to and from the APU may be handled by the associated processor using conventional programmed I/O, or may be handled by a direct memory access controller for improved performance. Upon completion of each command, the APU issues an end of execution signal that may be used as an interrupt by the CPU to help coordinate program execution.

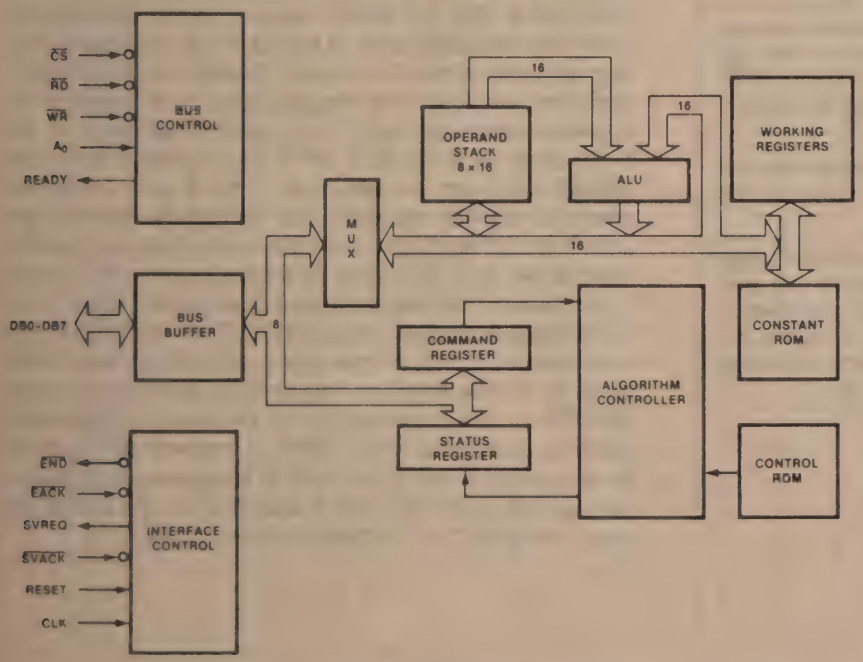


Figure 1. Block Diagram

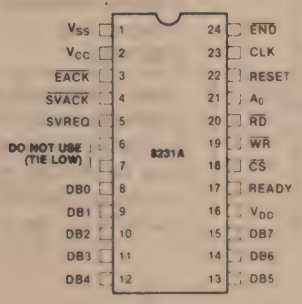


Figure 2. Pin Configuration

Order Number: 231305-001

Table 1. Pin Description

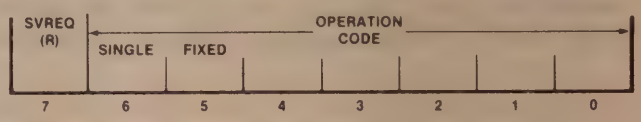
Symbol	Pin No.	Type	Name and Function
V_{CC}	2		**Power:** +5 Volt power supply.
V_{DD}	16		**Power:** +12 Volt power supply.
V_{SS}	1		**Ground.**
CLK	23	I	**Clock:** An external, TTL compatible, timing source is applied to the CLK pin.
RESET	22	I	**Reset:** The active high reset signal provides initialization for the chip. RESET also terminates any operation in progress. RESET clears the status register and places the 8231A into the idle state. Stack contents and command registers are not affected (5 clock cycles).
$\overline{CS}$	18	I	**Chip Select:** $\overline{CS}$ is an active low input signal which selects the 8231A and enables communication with the data bus.
A_0	21	I	**Address:** In conjunction with the $\overline{RD}$ and WR signals, the A_0 control line establishes the type of communication that is to be performed with the 8231A as shown below:

A_0	$\overline{RD}$	$\overline{WR}$	Function
0	1	0	Enter data byte into stack
0	0	1	Read data byte from stack
1	1	0	Enter command
1	0	1	Read status

Symbol	Pin No.	Type	Name and Function
$\overline{RD}$	20	I	**Read:** This active low input indicates that data or status is to be read from the 8231A if CS is low.
$\overline{WR}$	19	I	**Write:** This active low input indicates that data or a command is to be written into the 8231A if CS is low.
$\overline{EACK}$	3	I	**End of Execution:** This active low input clears the end of execution output signal ($\overline{END}$). If $\overline{EACK}$ is tied low, the $\overline{END}$ output will be a pulse that is one clock period wide.
$\overline{SVACK}$	4	I	**Service Request:** This active low input clears the service request output (SVREQ).
$\overline{END}$	24	O	**End:** This active low, open-drain output indicates that execution of the previously entered command is complete. It can be used as an interrupt request and is cleared by $\overline{EACK}$, RESET or any read or write access to the 8231.
SVREQ	5	O	**Service Request:** This active high output signal indicates that command execution is complete and that post execution service was requested in the previous command byte. It is cleared by $\overline{SVACK}$, the next command output to the device, or by RESET.
READY	17	O	**Ready:** This active high output indicates that the 8231A is able to accept communication with the data bus. When an attempt is made to read data, write data or to enter a new command while the 8231A is executing a command, READY goes low until execution of the current command is complete (See READY Operation, p. 5).
DB0-DB7	8-15	I/O	**Data Bus:** These eight bidirectional lines provide for transfer of commands, status and data between the 8231A and the CPU. The 8231A can drive the data bus only when CS and RD are low.

COMMAND STRUCTURE

Each command entered into the 8231A consists of a single 8-bit byte having the format illustrated below:



Bits 0-4 select the operation to be performed as shown in the table. Bits 5-6 select the data format appropriate to the selected operation. If bit 5 is a 1, a fixed point data format is specified. If bit 5 is a 0, floating point format is specified. Bit 6 selects the precision of the data to be operated upon by fixed point commands only (if bit 5 = 0, bit 6 must be 0). If bit 6 is a 1, single-precision (16-bit) operands are assumed. If bit 6 is a 0, double-precision (32-bit) operands are indicated. Results are undefined for all illegal combinations of bits in the command byte. Bit 7 indicates whether a service request is to be issued after the command is executed. If bit 7 is a 1, the service request output (SVREQ) will go high at the conclusion of the command and will remain high until reset by a low level on the service acknowledge pin ($\overline{SVACK}$) or until completion of execution of the succeeding command where service request (bit 7) is 0. Each command issued to the 8231A requests post execution service based upon the state of bit 7 in the command byte. When bit 7 is a 0, SVREQ remains low.

Table 2. 32-Bit Floating Point Instructions

Instruction	Description	Hex[1] Code	Stack Contents[2] After Execution A B C D	Status Flags[4] Affected
ACOS	Inverse Cosine of A	0 6	R U U U	S, Z, E
ASIN	Inverse Sine of A	0 5	R U U U	S, Z, E
ATAN	Inverse Tangent of A	0 7	R B U U	S, Z
CHSF	Sign Change of A	1 5	R B C D	S, Z
COS	Cosine of A (radians)	0 3	R B U U	S, Z
EXP	e^A Function	0 A	R B U U	S, Z, E
FADD	Add A and B	1 0	R C D U	S, Z, E
FDIV	Divide B by A	1 3	R C D U	S, Z, E
FLTD	32-Bit Integer to Floating Point Conversion	1 C	R B C U	S, Z
FLTS	16-Bit Integer to Floating Point Conversion	1 D	R B C U	S, Z
FMUL	Multiply A and B	1 2	R C D U	S, Z, E
FSUB	Subtract A from B	1 1	R C D U	S, Z, E
LOG	Common Logarithm (base 10) of A	0 8	R B U U	S, Z, E
LN	Natural Logarithm of A	0 9	R B U U	S, Z, E
POPF	Stack Pop	1 8	B C D A	S, Z
PTOF	Stack Push	1 7	A A B C	S, Z
PUPI	Push π onto Stack	1 A	R A B C	S, Z
PWR	B^A Power Function	0 B	R C U U	S, Z, E
SIN	Sine of A (radians)	0 2	R B U U	S, Z
SQRT	Square Root of A	0 1	R B C U	S, Z, E
TAN	Tangent of A (radians)	0 4	R B U U	S, Z, E
XCHF	Exchange A and B	1 9	B A C D	S, Z

Table 3. 32-Bit Integer Instructions

Instruction	Description	Hex[1] Code	Stack Contents[2] After Execution A B C D	Status Flags[4] Affected
CHSD	Sign Change of A	3 4	R B C D	S, Z, O
DADD	Add A and B	2 C	R C D A	S, Z, C, E
DDIV	Divide B by A	2 F	R C D U	S, Z, E
DMUL	Multiply A and B (R = lower 32-bits)	2 E	R C D U	S, Z, O
DMUU	Multiply A and B (R = upper 32-bits)	3 6	R C D U	S, Z, O
DSUB	Subtract A from B	2 D	R C D A	S, Z, C, O
FIXD	Floating Point to Integer Conversion	1 E	R B C U	S, Z, O
POPD	Stack Pop	3 8	B C D A	S, Z
PTOD	Stack Push	3 7	A A B C	S, Z
XCHD	Exchange A and B	3 9	B A C D	S, Z

Table 4. 16-Bit Integer Instructions

Instruction	Description	Hex[1] Code	Stack Contents[3] After Execution A_U A_L B_U B_L C_U C_L D_U D_L	Status Flags[4] Affected
CHSS	Change Sign of A_U	7 4	R A_L B_U B_L C_U C_L D_U D_L	S, Z, O
FIXS	Floating Point to Integer Conversion	1 F	R B_U B_L C_U C_L U U U	S, Z, O
POPS	Stack Pop	7 8	A_L B_U B_L C_U C_L D_U D_L A_U	S, Z
PTOS	Stack Push	7 7	A_U A_U A_L B_U B_L C_U C_L D_U	S, Z
SADD	Add A_U and A_L	6 C	R B_U B_L C_U C_L D_U D_L A_U	S, Z, C, E
SDIV	Divide A_L by A_U	6 F	R B_U B_L C_U C_L D_U D_L U	S, Z, E
SMUL	Multiply A_L by A_U (R = lower 16-bits)	6 E	R B_U B_L C_U C_L D_U D_L U	S, Z, E
SMUU	Multiply A_L by A_U (R = upper 16-bits)	7 6	R B_U B_L C_U C_L D_U D_L U	S, Z, E
SSUB	Subtract A_U from A_L	6 D	R B_U B_L C_U C_L D_U D_L A_U	S, Z, C, E
XCHS	Exchange A_U and A_L	7 9	A_L A_U B_U B_L C_U C_L D_U D_L	S, Z
NOP	No Operation	0 0	A_U A_L B_U B_L C_U C_L D_U D_L	

Notes: 1. In the hex code column, SVREQ is a 0.
2. The stack initially is composed of four 32-bit numbers (A, B, C, D). A is equivalent to Top Of Stack (TOS) and B is Next On Stack (NOS). Upon completion of a command the stack is composed of: the result (R); undefined (U); or the initial contents (A, B, C, or D).
3. The stack initially is composed of eight 16-bit numbers (A_U, A_L, B_U, B_L, C_U, C_L, D_U, D_L). A_U is the TOS and A_L is NOS. Upon completion of a command the stack is composed of: the result (R); undefined (U); or the initial contents (A_U, A_L, B_U, B_L, . . .).
4. Nomenclature: Sign (S); Zero (Z); Overflow (O); Carry (C); Error Code Field (E).

DATA FORMATS

The 8231A arithmetic processing unit handles operands in both fixed point and floating point formats. Fixed point operands may be represented in either single (16-bit operands) or double precision (32-bit operands), and are always represented as binary, two's complement values.

SINGLE PRECISION FIXED POINT FORMAT

DOUBLE PRECISION FIXED POINT FORMAT

The sign (positive or negative) of the operand is located in the most significant bit (MSB). Positive values are represented by a sign bit of zero (S = 0). Negative values are represented by the two's complement of the corresponding positive value with a sign bit equal to 1 (S = 1). The range of values that may be accommodated by each of these formats is −32,768 to +32,767 for single precision and −2,147,483,648 to +2,147,483,647 for double precision.

Floating point binary values are represented in a format that permits arithmetic to be performed in a fashion analogous to operations with decimal values expressed in scientific notation.

$$(5.83 \times 10^2)\,(8.16 \times 10^1) = (4.75728 \times 10^4)$$

In the decimal system, data may be expressed as values between 0 and 10 times 10 raised to a power that effectively shifts the implied decimal point right or left the number of places necessary to express the result in conventional form (e.g., 47,572.8). The value-portion of the data is called the mantissa. The exponent may be either negative or positive.

The concept of floating point notation has both a gain and a loss associated with it. The gain is the ability to represent the significant digits of data with values spanning a large dynamic range limited only by the capacity of the exponent field. For example, in decimal notation if the exponent field is two digits wide, and the mantissa is five digits, a range of values (positive or negative) from 1.0000×10^{-99} to $9.9999 \times 10^{+99}$ can be accommodated. The loss is that only the significant digits of the value can be represented. Thus there is no distinction in this representation between the values 123451 and 123452, for example, since each would be expressed as: 1.2345×10^5. The sixth digit has been discarded. In most applications where the dynamic range of values to be represented is large, the loss of significance, and hence accuracy of results, is a minor consideration. For greater precision a fixed point format could be chosen, although with a loss of potential dynamic range.

The 8231A is a binary arithmetic processor and requires that floating point data be represented by a fractional mantissa value between .5 and 1 multiplied by 2 raised to an appropriate power. This is expressed as follows:

$$\text{value} = \text{mantissa} \times 2^{\text{exponent}}$$

For example, the value 100.5 expressed in this form is $0.1100\ 1001 \times 2^7$. The decimal equivalent of this value may be computed by summing the components (powers of two) of the mantissa and then multiplying by the exponent as shown below:

$$\begin{aligned}\text{value} &= (2^{-1} + 2^{-2} + 2^{-5} + 2^{-8}) \times 2^7 \\ &= 0.5 + 0.25 + 0.03125 + 0.00290625) \times 128 \\ &= 0.78515625 \times 128 \\ &= 100.5\end{aligned}$$

FLOATING POINT FORMAT

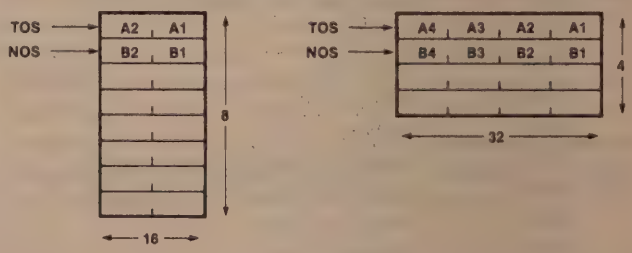
The format for floating point values in the 8231A is given below. The mantissa is expressed as a 24-bit (fractional) value; the exponent is expressed as a two's complement 7-bit value having a range of −64 to +63. The most significant bit is the sign of the mantissa (0 = positive, 1 = negative), for a total of 32 bits. The binary point is assumed to be to the left of the most significant mantissa bit (bit 23). All floating point data values must be normalized. Bit 23 must be equal to 1, except for the value zero, which is represented by all zeros.

The range of values that can be represented in this format is $\pm(2.7 \times 10^{-20}$ to $9.2 \times 10^{18})$ and zero.

FUNCTIONAL DESCRIPTION

STACK CONTROL

The user interface to the 8231A includes access to an 8 level 16-bit wide data stack. Since single precision fixed point operands are 16-bits in length, eight such values may be maintained in the stack. When using double precision fixed point or floating point formats four values may be stored. The stack in these two configurations can be visualized as shown below:



Data are written onto the stack, eight bits at a time, in the order shown (A1, A2, A3, . . .). Data are removed from the stack in reverse byte order (A4, A3, A2 . . .). Data should be entered onto the stack in multiples of the number of bytes appropriate to the chosen data format.

DATA ENTRY

Data entry is accomplished by bringing the chip select ($\overline{CS}$), the command/data line (A_0), and $\overline{WR}$ low, as shown in the timing diagram. The entry of each new data word "pushes down" the previously entered data and places the new byte on the top of stack (TOS). Data on the bottom of the stack prior to a stack entry are lost.

DATA REMOVAL

Data are removed from the stack in the 8231A by bringing chip select ($\overline{CS}$), command/data (A_0), and $\overline{RD}$ low as shown in the timing diagram. The removal of each data word redefines TOS so that the next successive byte to be removed becomes TOS. Data removed from the stack rotates to the bottom of the stack.

COMMAND ENTRY

After the appropriate number of bytes of data have been entered onto the stack, a command may be issued to perform an operation on that data. Commands which require two operands for execution (e.g., add) operate on the TOS and NOS values. Single operand commands operate only on the TOS.

Commands are issued to the 8231A by bringing the chip select ($\overline{CS}$) line low, command data (A_0) line high, and $\overline{WR}$ line low as indicated by the timing diagram. After a command is issued, the CPU can continue execution of its program concurrently with the 8231A command execution.

COMMAND COMPLETION

The 8231A signals the completion of each command execution by lowering the End Execution line ($\overline{END}$). Simultaneously, the busy bit in the status register is cleared and the Service Request bit of the command register is checked. If it is a "1" the service request output level (SVREQ) is raised. $\overline{END}$ is cleared on receipt of an active low End Acknowledge ($\overline{EACK}$) pulse. Similarly, the service request line is cleared by recognition of an active low Service Acknowledge ($\overline{SVACK}$) pulse.

READY OPERATION

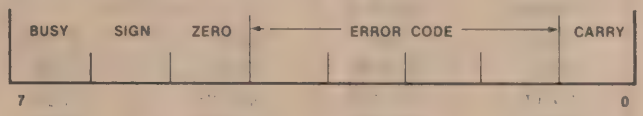
An active high ready (READY) is provided. This line is high in its quiescent state and is pulled low by the 8231A under the following conditions:

1. A previously initiated operation is in progress (device busy) and Command Entry has been attempted. In this case, the READY line will be pulled low and remain low until completion of the current command execution. It will then go high, permitting entry of the new command.
2. A previously initiated operation is in progress and stack access has been attempted. In this case, the READY line will be pulled low, will remain in that state until execution is complete, and will then be raised to permit completion of the stack access.
3. The 8231A is not busy, and data removal has been requested. READY will be pulled low for the length of time necessary to transfer the byte from the top of stack to the interface latch, and will then go high, indicating availability of the data.
4. The 8231A is not busy, and a data entry has been requested. READY will be pulled low for the length of time required to ascertain if the preceding data byte, if any, has been written to the stack. If so READY will immediately go high. If not, READY will remain low until the interface latch is free and will then go high.
5. When a status read has been requested, READY will be pulled low for the length of time necessary to transfer the status to the interface latch, and will then be raised to permit completion of the status read. Status may be read whether or not the 8231A is busy.

When READY goes low, the APU expects the bus control signals present at the time to remain stable until READY goes high.

DEVICE STATUS

Device status is provided by means of an internal status register whose format is shown below:



BUSY: Indicates that 8231A is currently executing a command (1 = Busy)

SIGN: Indicates that the value on the top of stack is negative (1 = Negative)

ZERO: Indicates that the value on the top of stack is zero (1 = Value is zero)

ERROR CODE: This field contains an indication of the validity of the result of the last operation. The error codes are:

- 0000 — No error
- 1000 — Divide by zero
- 0100 — Square root or log of negative number
- 1100 — Argument of inverse sine, cosine, or e^x too large
- XX10 — Underflow
- XX01 — Overflow

CARRY: Previous operation resulted in carry or borrow from most significant bit. (1 = Carry/Borrow, 0 = No Carry/No Borrow.)

If the BUSY bit in the status register is a one, the other status bits are not defined; if zero, indicating not busy, the operation is complete and the other status bits are defined as given above.

READ STATUS

The 8231A status register can be read by the CPU at any time (whether an operation is in progress or not) by bringing the chip select ($\overline{CS}$) low, the command/data line (A_0) high, and lowering $\overline{RD}$. The status register is then gated onto the data bus and may be input by the CPU.

EXECUTION TIMES

Timing for execution of the 8231A command set is contained below. All times are given in terms of clock cycles. Where substantial variation of execution times

is possible, the minimum and maximum values are quoted; otherwise, typical values are given. Variations are data dependent.

Total execution times may require allowances for operand transfer into the APU, command execution, and result retrieval from the APU. Except for command execution, these times will be heavily influenced by the nature of the data, the control interface used, the speed of memory, the CPU used, the priority allotted to DMA and Interrupt operations, the size and number of operands to be transferred, and the use of chained calculations, etc.

Table 5. Command Execution Times

Command Mnemonic	Clock Cycles	Command Mnemonic	Clock Cycles	Command Mnemonic	Clock Cycles	Command Mnemonic	Clock Cycles
SADD	17	FADD	54-368	LN	4298-6956	POPF	12
SSUB	30	FSUB	70-370	EXP	3794-4878	XCHS	18
SMUL	84-94	FMUL	146-168	PWR	8290-12032	XCHD	26
SMUU	80-98						
SDIV	84-94	FDIV	154-184	NOP	4	XCHF	26
DADD	21	SORT	800	CHSS	23	PUPI	16
DSUB	38	SIN	4464	CHSD	27		
DMUL	194-210	COS	4118	CHSF	18		
DMUU	182-218						
DDIV	208	TAN	5754	PTOS	16		
FIXS	92-216	ASIN	7668	PTOD	20		
FIXD	100-346	ACOS	7734	PTOF	20		
FLTS	98-186	ATAN	6006	POPS	10		
FLTD	98-378	LOG	4474-7132	POPD	12		

DERIVED FUNCTION DISCUSSION

Computer approximations of transcendental functions are often based on some form of polynomial equation, such as:

$$F(X) = A_0 + A_1X + A_2X^2 + A_3X^3 + A_4X^4 \ldots \quad (1\text{-}1)$$

The primary shortcoming of an approximation in this form is that it typically exhibits very large errors when the magnitude of $|X|$ is large, although the errors are small when $|X|$ is small. With polynomials in this form, the error distribution is markedly uneven over any arbitrary interval.

A set of approximating functions exists that not only minimizes the maximum error but also provides an even distribution of errors within the selected data representation interval. These are known as Chebyshev Polynomials and are are based upon cosine functions. These functions are defined as follows:

$$T_n(X) = \text{Cos } n\theta; \text{ where } n = 0,1,2 \ldots \quad (1\text{-}2)$$
$$\theta = \text{Cos}^{-1}X$$

The various terms of the Chebyshev series can be computed as shown below:

$$T_0(X) = \text{Cos}(0 \cdot \theta) = \text{Cos}(0) = 1 \quad (1\text{-}4)$$
$$T_1(X) = \text{Cos}(\text{Cos}^{-1}X) = X \quad (1\text{-}5)$$
$$T_2(X) = \text{Cos } 2\theta = 2\text{Cos}^2\theta - 1 = 2\text{Cos}^2(\text{Cos}^{-1}X) - 1 \quad (1\text{-}6)$$
$$= 2X^2 - 1$$

In general, the next term in the Chebyshev series can be recursively derived from the previous term as follows:

$$T_n(X) = 2X[T_n - 1(X)] - T_n - 2(X); n \geq 2 \quad (1\text{-}7)$$

Common logarithms are computed by multiplication of the natural logarithm by the conversion factor 0.43429448 and the error function is therefore the same as that for natural logarithm. The power function is realized by combination of natural log and exponential functions according to the equation:

$$X^Y = e^{y\text{Ln}x}.$$

The error for the power function is a combination of that for the logarithm and exponential functions.

Each of the derived functions is an approximation of the true function. Thus the result of a derived function will have an error. The absolute error is the difference between the function's result and the true result. A more useful measure of the function's error is relative error (absolute error/true result). This gives a measurement of the significant digits of algorithm accuracy. For the derived functions except LN, LOG, and PWR the relative error is typically 4×10^{-7}. For PWR the relative error is the summation of the EXP and LN errors, 7×10^{-7}. For LN and LOG, the absolute error is 2×10^{-7}.

APPLICATION INFORMATION

The diagram in Figure 4 shows the interface connections for the APU with operand transfers handled by an 8237 DMA controller, and CPU coordination handled by an Interrupt Controller. The APU interrupts the CPU to indicate that a command has been completed. When the performance enhancements provided by the DMA and Interrupt operations are not required, the APU interface can be simplified as shown in Figure 3. The 8231A APU is designed with a general purpose 8-bit data bus and interface control so that it can be conveniently used with any general 8-bit processor.

In many systems it will be convenient to use the microcomputer system clock to drive the APU clock input. In the case of 8080A systems it would be the ϕ2TTL signal. Its cycle time will usually fall in the range of 250 ns to 1000 ns, depending on the system speed.

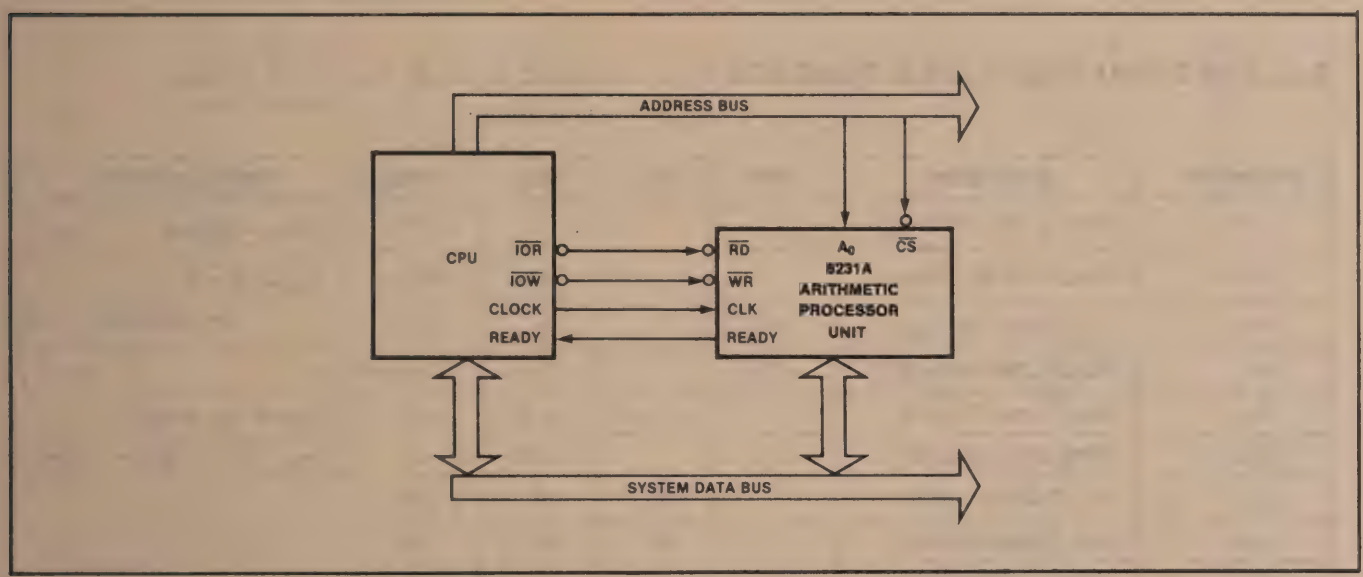


Figure 3. Minimum Configuration Example

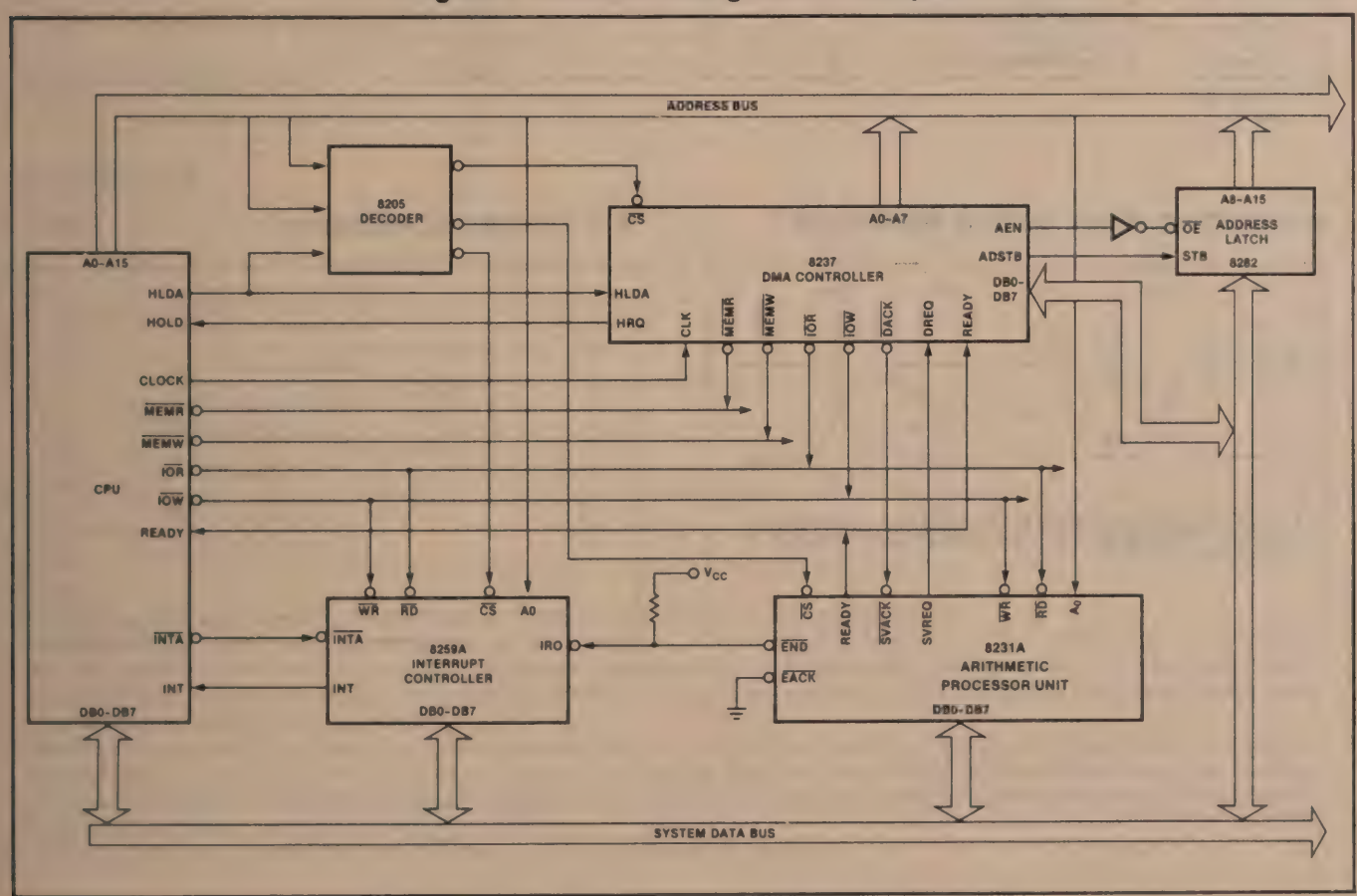


Figure 4. High Performance Configuration Example

ABSOLUTE MAXIMUM RATINGS*

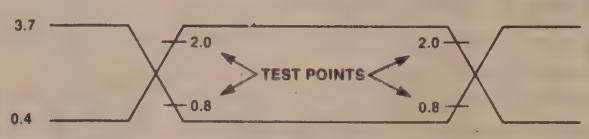
Storage Temperature . . . . . – 65°C to + 150°C
Ambient Temperature Under Bias . . . . . 0°C to 70°C
V_{DD} with Respect to V_{SS} . . . . . – 0.5V to + 15.0V
V_{CC} with Respect to V_{SS} . . . . . – 0.5V to + 7.0V
All Signal Voltages with Respect
to V_{SS} . . . . . – 0.5V to + 7.0V
Power Dissipation 2.0W

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may effect device reliability.*

D.C. AND OPERATING CHARACTERISTICS (T_A = 0°C to 70°C, V_{SS} = 0V, V_{CC} = +5V ± 10%, V_{DD} = +12V ± 10%)

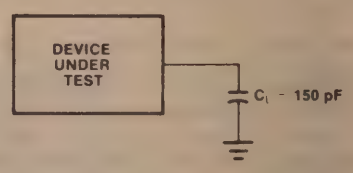
Parameters	Description	Min.	Typ.	Max.	Units	Test Conditions
V_{OH}	Output HIGH Voltage	3.7			Volts	I_{OH} = – 200 μA
V_{OL}	Output LOW Voltage			0.4	Volts	I_{OL} = 3.2 mA
V_{IH}	Input HIGH Voltage	2.0		V_{CC}	Volts	
V_{IL}	Input LOW Voltage	– 0.5		0.8	Volts	
I_{IL}	Input Load Current			± 10	μA	$V_{SS} \leq V_{IN} \leq V_{CC}$
I_{OFL}	Data Bus Leakage			± 10	μA	V_{SS} +0.45 ≤ V_{OUT} ≤ V_{CC}
I_{CC}	V_{CC} Supply Current		50	95	mA	
I_{DD}	V_{DD} Supply Current		50	95	mA	
C_O	Output Capacitance		8		pF	fc = 1.0 MHz, Inputs = 0V
C_I	Input Capacitance		5		pF	
C_{IO}	I/O Capacitance		10		pF	

A.C. TESTING INPUT, OUTPUT WAVEFORM



A.C. TESTING: INPUTS ARE DRIVEN AT 3.7V FOR A LOGIC "1" AND 0.4V FOR A LOGIC "0." TIMING MEASUREMENTS ARE MADE AT 2.0V FOR A LOGIC "1" AND 0.8V FOR A LOGIC "0."

A.C. TESTING LOAD CIRCUIT



231305-001

A.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{SS} = 0V, V_{CC} = +5V ± 10%, V_{DD} = +12V ± 10%)

READ OPERATION

Symbol	Parameter		8231A-8 Min.	8231A-8 Max.	8231A Min.	8231A Max.	Units
t_{AR}	A_0, $\overline{CS}$ Setup to $\overline{RD}$		0		0		ns
t_{RA}	A_0, $\overline{CS}$ Hold from $\overline{RD}$		0		0		ns
t_{RY}	READY ↓ from $\overline{RD}$ ↓ Delay (Note 2)			150		100	ns
t_{YR}	READY ↑ to $\overline{RD}$ ↑		0		0		ns
t_{RRR}	READY Pulse Width (Note 3)	Data	3.5 t_{CY} + 50		3.5 t_{CY} + 50		ns
		Status	1.5 t_{CY} + 50		1.5 t_{CY} + 50		ns
t_{RDE}	Data Bus Enable from $\overline{RD}$ ↓		50		50		ns
t_{DRY}	Data Valid to READY ↑		0		0		ns
t_{DF}	Data Float after $\overline{RD}$ ↑		50	200	50	100	ns

WRITE OPERATION

Symbol	Parameter		8231A-8 Min.	8231A-8 Max.	8231A Min.	8231A Max.	Units
t_{AW}	A_0, $\overline{CS}$ Setup to $\overline{WR}$		0		0		ns
t_{WA}	A_0, $\overline{CS}$ Hold after $\overline{WR}$		60		25		ns
t_{WY}	READY ↓ from $\overline{WR}$ ↓ Delay (Note 2)			150		100	ns
t_{YW}	READY ↑ to $\overline{WR}$ ↑		0		0		ns
t_{RRW}	READY Pulse Width (Note 4)			50		50	ns
t_{WI}	Write Inactive Time (Note 4)	Command	4 t_{CY}		4 t_{CY}		ns
		Data	5 t_{CY}		5 t_{CY}		ns
t_{DW}	Data Setup to $\overline{WR}$		150		100		ns
t_{WD}	Data Hold after $\overline{WR}$		20		20		ns

OTHER TIMINGS

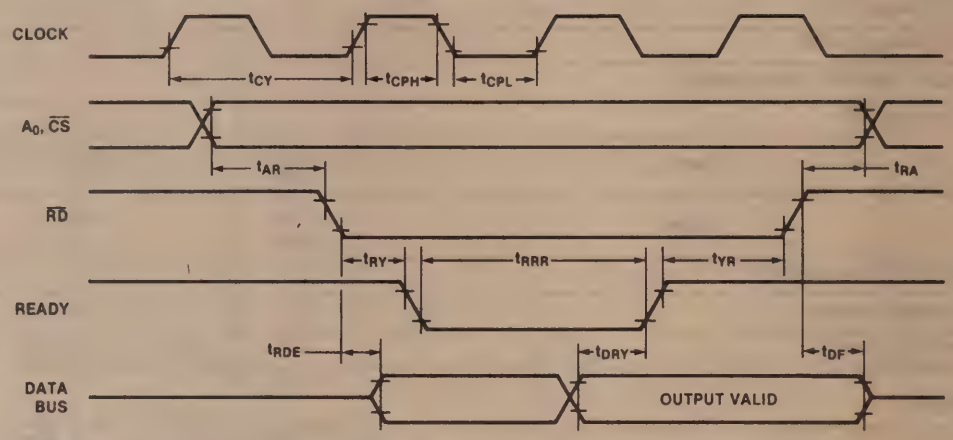
Symbol	Parameter	8231A-8 Min.	8231A-8 Max.	8231A Min.	8231A Max.	Units
t_{CY}	Clock Period	480	5000	250	2500	ns
t_{CPH}	Clock Pulse High Width	200		100		ns
t_{CPL}	Clock Pulse Low Width	240		120		ns
t_{EE}	$\overline{END}$ Pulse Width (Note 5)	400		200		ns
t_{EAE}	$\overline{EACK}$ ↓ to $\overline{END}$ ↑ Delay		200		150	ns
t_{AA}	$\overline{EACK}$ Pulse Width	100		50		ns
t_{SA}	$\overline{SVACK}$ ↓ to SVREQ ↓ Delay		300		150	ns
t_{SS}	$\overline{SVACK}$ Pulse Width	100		50		ns

NOTES:

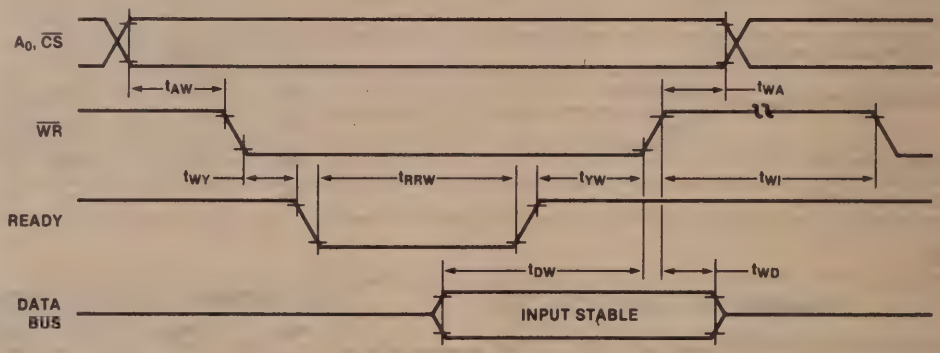
1. Typical values are for T_A=25°C, nominal supply voltages and nominal processing parameters.
2. READY is pulled low for both command and data operations.
3. Minimum values shown assume no previously entered command is being executed for the data access. If a previously entered command is being executed, READY low pulse width is the time to complete execution plus the time shown. Status may be read at any time without exceeding the time shown.
4. READY low pulse width is less than 50 ns when writing into the data port or the control port as long as the duty cycle requirement (t_{WI}) is observed and no previous command is being executed. t_{WI} may be safely violated as long as the extended t_{RRW} that results is observed. If a previously entered command is being executed, READY low pulse width is the time to complete execution plus the time shown. These timings refer specifically to the 8231A.
5. $\overline{END}$ low pulse width is specified for $\overline{EACK}$ tied to VSS. Otherwise t_{EAE} applies.

WAVEFORMS

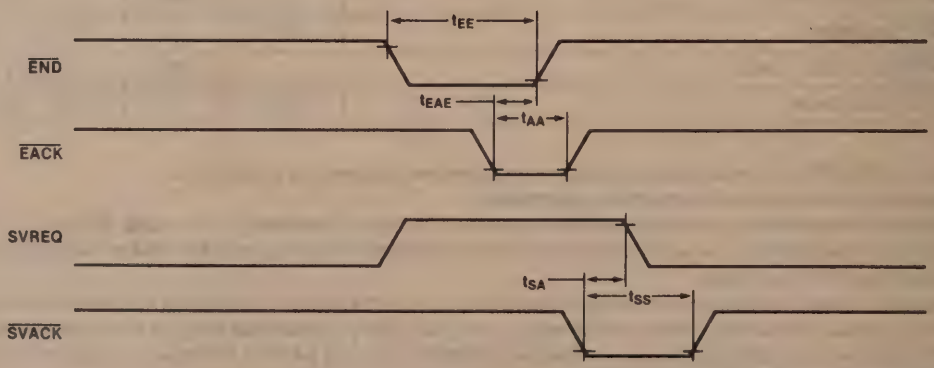
READ OPERATION



WRITE OPERATION



INTERRUPT OPERATION



8253/8253-5
PROGRAMMABLE INTERVAL TIMER

- **MCS-85™ Compatible 8253-5**
- **3 Independent 16-Bit Counters**
- **DC to 2.6 MHz**
- **Programmable Counter Modes**
- **Count Binary or BCD**
- **Single +5V Supply**
- **Available in EXPRESS**
  —Standard Temperature Range
  —Extended Temperature Range

The Intel® 8253 is a programmable counter/timer device designed for use as an Intel microcomputer peripheral. It uses nMOS technology with a single +5V supply and is packaged in a 24-pin plastic DIP.

It is organized as 3 independent 16-bit counters, each with a count rate of up to 2.6 MHz. All modes of operation are software programmable.

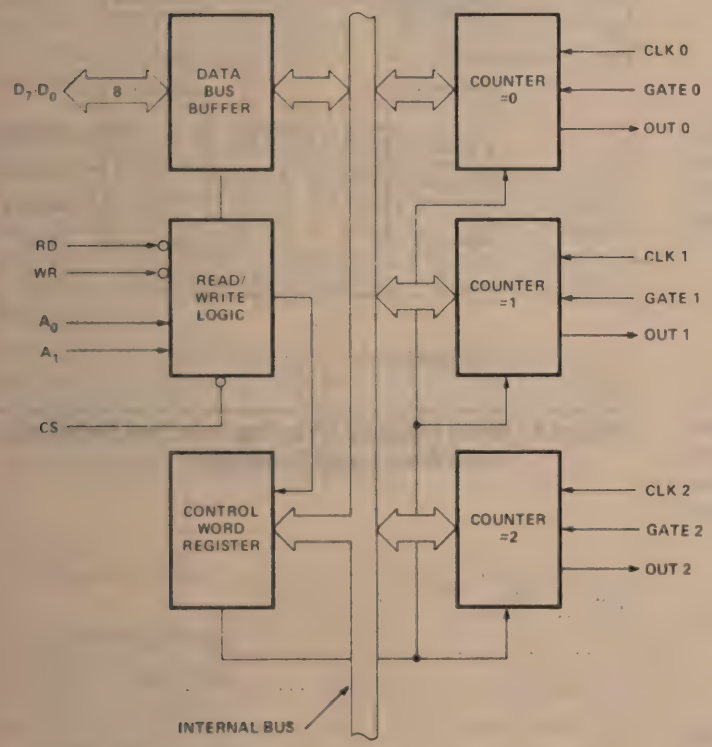


Figure 1. Block Diagram

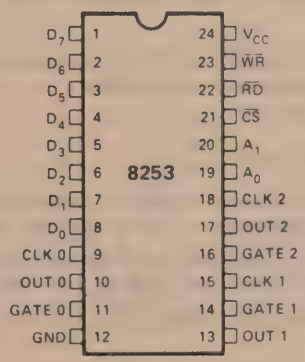


Figure 2. Pin Configuration

©INTEL CORPORATION, 1983

Order Number: 231306-001

FUNCTIONAL DESCRIPTION

General

The 8253 is a programmable interval timer/counter specifically designed for use with the Intel™ Microcomputer systems. Its function is that of a general purpose, multi-timing element that can be treated as an array of I/O ports in the system software.

The 8253 solves one of the most common problems in any microcomputer system, the generation of accurate time delays under software control. Instead of setting up timing loops in systems software, the programmer configures the 8253 to match his requirements, initializes one of the counters of the 8253 with the desired quantity, then upon command the 8253 will count out the delay and interrupt the CPU when it has completed its tasks. It is easy to see that the software overhead is minimal and that multiple delays can easily be maintained by assignment of priority levels.

Other counter/timer functions that are non-delay in nature but also common to most microcomputers can be implemented with the 8253.

- Programmable Rate Generator
- Event Counter
- Binary Rate Multiplier
- Real Time Clock
- Digital One-Shot
- Complex Motor Controller

Data Bus Buffer

This 3-state, bi-directional, 8-bit buffer is used to interface the 8253 to the system data bus. Data is transmitted or received by the buffer upon execution of INput or OUTput CPU instructions. The Data Bus Buffer has three basic functions.

1. Programming the MODES of the 8253.
2. Loading the count registers.
3. Reading the count values.

Read/Write Logic

The Read/Write Logic accepts inputs from the system bus and in turn generates control signals for overall device operation. It is enabled or disabled by CS so that no operation can occur to change the function unless the device has been selected by the system logic.

$\overline{RD}$ (Read)

A "low" on this input informs the 8253 that the CPU is inputting data in the form of a counters value.

$\overline{WR}$ (Write)

A "low" on this input informs the 8253 that the CPU is outputting data in the form of mode information or loading counters.

A0, A1

These inputs are normally connected to the address bus. Their function is to select one of the three counters to be operated on and to address the control word register for mode selection.

$\overline{CS}$ (Chip Select)

A "low" on this input enables the 8253. No reading or writing will occur unless the device is selected. The $\overline{CS}$ input has no effect upon the actual operation of the counters.

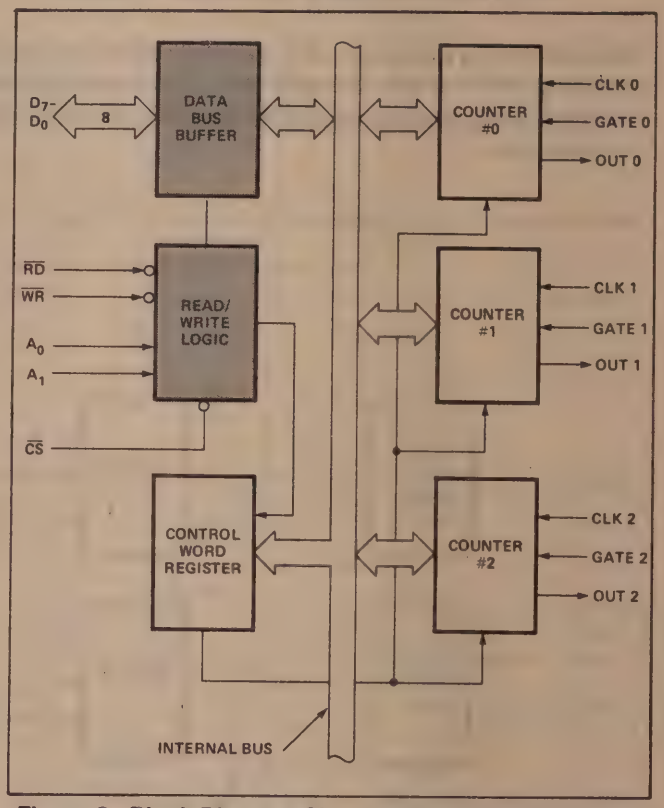


Figure 3. Block Diagram Showing Data Bus Buffer and Read/Write Logic Functions

$\overline{CS}$	$\overline{RD}$	$\overline{WR}$	A_1	A_0	
0	1	0	0	0	Load Counter No. 0
0	1	0	0	1	Load Counter No. 1
0	1	0	1	0	Load Counter No. 2
0	1	0	1	1	Write Mode Word
0	0	1	0	0	Read Counter No. 0
0	0	1	0	1	Read Counter No. 1
0	0	1	1	0	Read Counter No. 2
0	0	1	1	1	No-Operation 3-State
1	X	X	X	X	Disable 3-State
0	1	1	X	X	No-Operation 3-State

Control Word Register

The Control Word Register is selected when A0, A1 are 11. It then accepts information from the data bus buffer and stores it in a register. The information stored in this register controls the operational MODE of each counter, selection of binary or BCD counting and the loading of each count register.

The Control Word Register can only be written into; no read operation of its contents is available.

Counter #0, Counter #1, Counter #2

These three functional blocks are identical in operation so only a single Counter will be described. Each Counter consists of a single, 16-bit, pre-settable, DOWN counter. The counter can operate in either binary or BCD and its input, gate and output are configured by the selection of MODES stored in the Control Word Register.

The counters are fully independent and each can have separate Mode configuration and counting operation, binary or BCD. Also, there are special features in the control word that handle the loading of the count value so that software overhead can be minimized for these functions.

The reading of the contents of each counter is available to the programmer with simple READ operations for event counting applications and special commands and logic are included in the 8253 so that the contents of each counter can be read "on the fly" without having to inhibit the clock input.

8253 SYSTEM INTERFACE

The 8253 is a component of the Intel™ Microcomputer Systems and interfaces in the same manner as all other peripherals of the family. It is treated by the systems software as an array of peripheral I/O ports; three are counters and the fourth is a control register for MODE programming.

Basically, the select inputs A0, A1 connect to the A0, A1 address bus signals of the CPU. The $\overline{CS}$ can be derived directly from the address bus using a linear select method. Or it can be connected to the output of a decoder, such as an Intel® 8205 for larger systems.

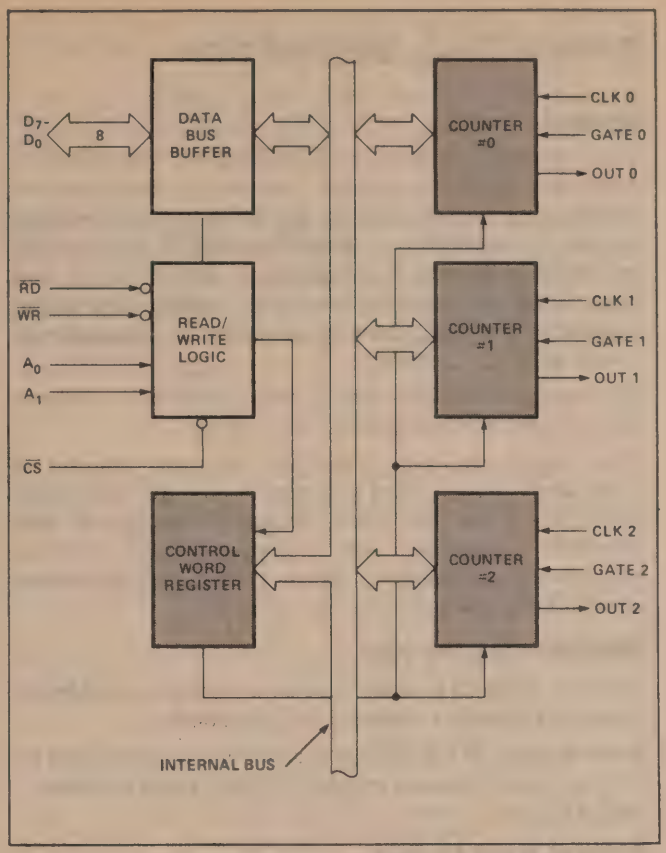


Figure 4. Block Diagram Showing Control Word Register and Counter Functions

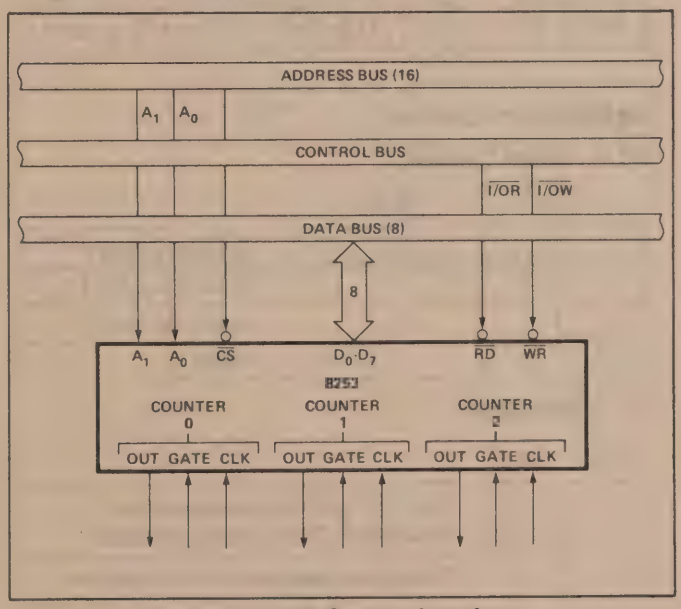


Figure 5. 8253 System Interface

OPERATIONAL DESCRIPTION

General

The complete functional definition of the 8253 is programmed by the systems software. A set of control words must be sent out by the CPU to initialize each counter of the 8253 with the desired MODE and quantity information. Prior to initialization, the MODE, count, and output of all counters is undefined. These control words program the MODE, Loading sequence and selection of binary or BCD counting.

Once programmed, the 8253 is ready to perform whatever timing tasks it is assigned to accomplish.

The actual counting operation of each counter is completely independent and additional logic is provided on-chip so that the usual problems associated with efficient monitoring and management of external, asynchronous events or rates to the microcomputer system have been eliminated.

Programming the 8253

All of the MODES for each counter are programmed by the systems software by simple I/O operations.

Each counter of the 8253 is individually programmed by writing a control word into the Control Word Register. (A0, A1 = 11)

Control Word Format

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SC1	SC0	RL1	RL0	M2	M1	M0	BCD

Definition of Control

SC — Select Counter:

SC1	SC0	
0	0	Select Counter 0
0	1	Select Counter 1
1	0	Select Counter 2
1	1	Illegal

RL — Read/Load:

RL1	RL0	
0	0	Counter Latching operation (see READ/WRITE Procedure Section)
1	0	Read/Load most significant byte only.
0	1	Read/Load least significant byte only.
1	1	Read/Load least significant byte first, then most significant byte.

M — MODE:

M2	M1	M0	
0	0	0	Mode 0
0	0	1	Mode 1
X	1	0	Mode 2
X	1	1	Mode 3
1	0	0	Mode 4
1	0	1	Mode 5

BCD:

0	Binary Counter 16-bits
1	Binary Coded Decimal (BCD) Counter (4 Decades)

Counter Loading

The count register is not loaded until the count value is written (one or two bytes, depending on the mode selected by the RL bits), followed by a rising edge and a falling edge of the clock. Any read of the counter prior to that falling clock edge may yield invalid data.

MODE Definition

MODE 0: Interrupt on Terminal Count. The output will be initially low after the mode set operation. After the count is loaded into the selected count register, the output will remain low and the counter will count. When terminal count is reached the output will go high and remain high until the selected count register is reloaded with the mode or a new count is loaded. The counter continues to decrement after terminal count has been reached.

Rewriting a counter register during counting results in the following:

(1) Write 1st byte stops the current counting.
(2) Write 2nd byte starts the new count.

MODE 1: Programmable One-Shot. The output will go low on the count following the rising edge of the gate input.

The output will go high on the terminal count. If a new count value is loaded while the output is low it will not affect the duration of the one-shot pulse until the succeeding trigger. The current count can be read at any time without affecting the one-shot pulse.

The one-shot is retriggerable, hence the output will remain low for the full count after any rising edge of the gate input.

MODE 2: Rate Generator. Divide by N counter. The output will be low for one period of the input clock. The period from one output pulse to the next equals the number of input counts in the count register. If the count register is reloaded between output pulses the present period will not be affected, but the subsequent period will reflect the new value.

The gate input, when low, will force the output high. When the gate input goes high, the counter will start from the initial count. Thus, the gate input can be used to synchronize the counter.

When this mode is set, the output will remain high until after the count register is loaded. The output then can also be synchronized by software.

MODE 3: Square Wave Rate Generator. Similar to MODE 2 except that the output will remain high until one half the count has been completed (for even numbers) and go low for the other half of the count. This is accomplished by decrementing the counter by two on the falling edge of each clock pulse. When the counter reaches terminal count, the state of the output is changed and the counter is reloaded with the full count and the whole process is repeated.

If the count is odd and the output is high, the first clock pulse (after the count is loaded) decrements the count by 1. Subsequent clock pulses decrement the clock by 2. After timeout, the output goes low and the full count is reloaded. The first clock pulse (following the reload) decrements the counter by 3. Subsequent clock pulses decrement the count by 2 until timeout. Then the whole process is repeated. In this way, if the count is odd, the output will be high for $(N+1)/2$ counts and low for $(N-1)/2$ counts.

In Modes 2 and 3, if a CLK source other than the system clock is used, GATE should be pulsed immediately following $\overline{WR}$ of a new count value.

MODE 4: Software Triggered Strobe. After the mode is set, the output will be high. When the count is loaded, the counter will begin counting. On terminal count, the output will go low for one input clock period, then will go high again.

If the count register is reloaded during counting, the new count will be loaded on the next CLK pulse. The count will be inhibited while the GATE input is low.

MODE 5: Hardware Triggered Strobe. The counter will start counting after the rising edge of the trigger input and will go low for one clock period when the terminal count is reached. The counter is retriggerable. The output will not go low until the full count after the rising edge of any trigger.

Modes \ Signal Status	Low Or Going Low	Rising	High
0	Disables counting	——	Enables counting
1	——	1) Initiates counting 2) Resets output after next clock	——
2	1) Disables counting 2) Sets output immediately high	1) Reloads counter 2) Initiates counting	Enables counting
3	1) Disables counting 2) Sets output immediately high	1) Reloads counter 2) Initiates counting	Enables counting
4	Disables counting	——	Enables counting
5	——	Initiates counting	——

Figure 6. Gate Pin Operations Summary

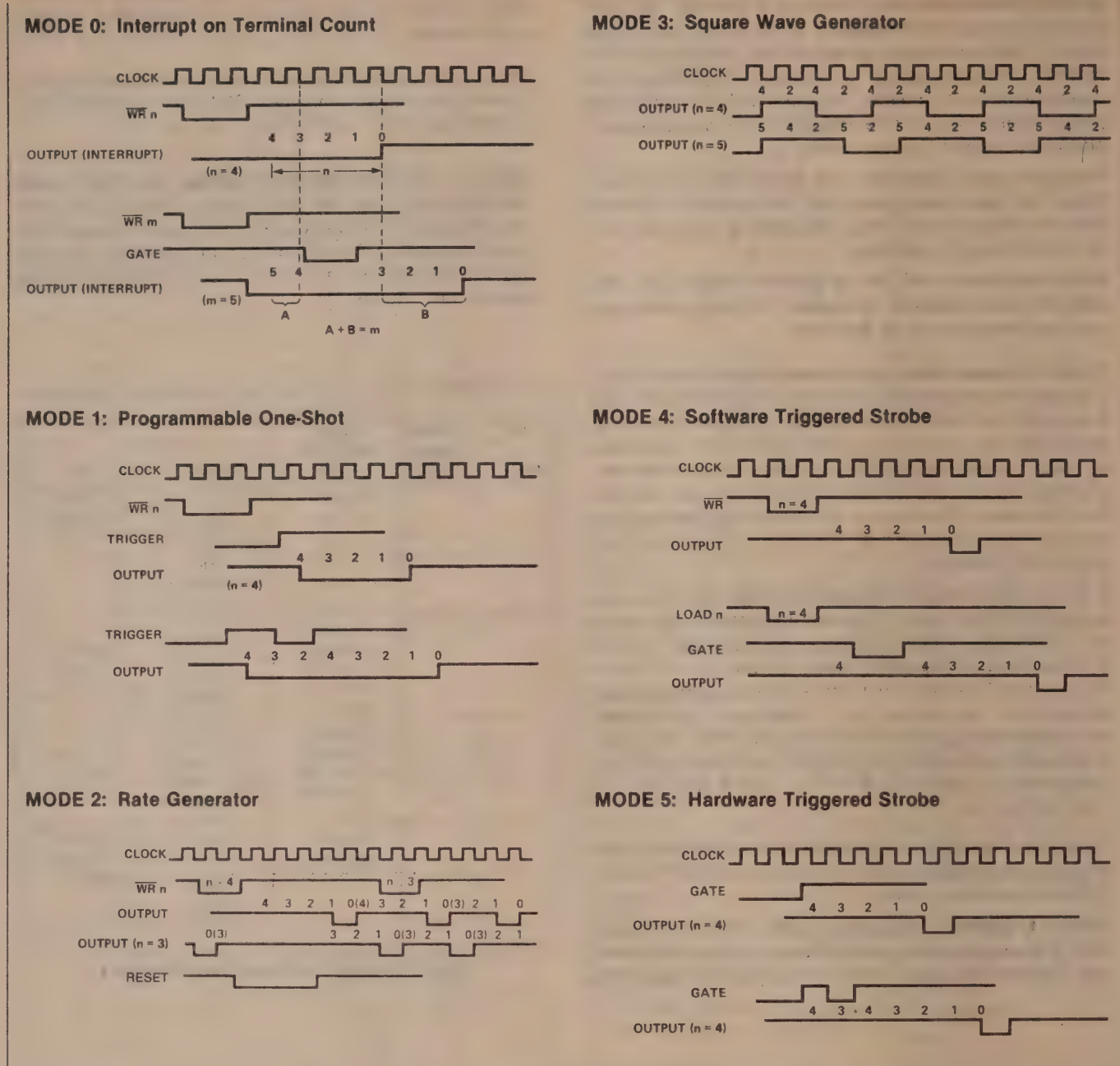


Figure 7. 8253 Timing Diagrams

8253 READ/WRITE PROCEDURE

Write Operations

The systems software must program each counter of the 8253 with the mode and quantity desired. The programmer must write out to the 8253 a MODE control word and the programmed number of count register bytes (1 or 2) prior to actually using the selected counter.

The actual order of the programming is quite flexible. Writing out of the MODE control word can be in any sequence of counter selection, e.g., counter #0 does not have to be first or counter #2 last. Each counter's MODE control word register has a separate address so that its loading is completely sequence independent. (SC0, SC1)

The loading of the Count Register with the actual count value, however, must be done in exactly the sequence programmed in the MODE control word (RL0, RL1). This loading of the counter's count register is still sequence independent like the MODE control word loading, but when a selected count register is to be loaded it must be loaded with the number of bytes programmed in the MODE control word (RL0, RL1). The one or two bytes to be loaded in the count register do not have to follow the associated MODE control word. They can be programmed at any time following the MODE control word loading as long as the correct number of bytes is loaded in order.

All counters are down counters. Thus, the value loaded into the count register will actually be decremented. Loading all zeroes into a count register will result in the maximum count (2^{16} for Binary or 10^4 for BCD). In MODE 0 the new count will not restart until the load has been completed. It will accept one of two bytes depending on how the MODE control words (RL0, RL1) are programmed. Then proceed with the restart operation.

	MODE Control Word Counter n
LSB	Count Register byte Counter n
MSB	Count Register byte Counter n

Note: Format shown is a simple example of loading the 8253 and does not imply that it is the only format that can be used.

Figure 8. Programming Format

			A1	A0
No. 1		MODE Control Word Counter 0	1	1
No. 2		MODE Control Word Counter 1	1	1
No. 3		MODE Control Word Counter 2	1	1
No. 4	LSB	Count Register Byte Counter 1	0	1
No. 5	MSB	Count Register Byte Counter 1	0	1
No. 6	LSB	Count Register Byte Counter 2	1	0
No. 7	MSB	Count Register Byte Counter 2	1	0
No. 8	LSB	Count Register Byte Counter 0	0	0
No. 9	MSB	Count Register Byte Counter 0	0	0

Note: The exclusive addresses of each counter's count register make the task of programming the 8253 a very simple matter, and maximum effective use of the device will result if this feature is fully utilized.

Figure 9. Alternate Programming Formats

Read Operations

In most counter applications it becomes necessary to read the value of the count in progress and make a computational decision based on this quantity. Event counters are probably the most common application that uses this function. The 8253 contains logic that will allow the programmer to easily read the contents of any of the three counters without disturbing the actual count in progress.

There are two methods that the programmer can use to read the value of the counters. The first method involves the use of simple I/O read operations of the selected counter. By controlling the A0, A1 inputs to the 8253 the programmer can select the counter to be read (remember that no read operation of the mode register is allowed A0, A1-11). The only requirement with this method is that in order to assure a stable count reading the actual operation of the selected counter must be inhibited either by controlling the Gate input or by external logic that inhibits the clock input. The contents of the counter selected will be available as follows:

first I/O Read contains the least significant byte (LSB).

second I/O Read contains the most significant byte (MSB).

Due to the internal logic of the 8253 it is absolutely necessary to complete the entire reading procedure. If two bytes are programmed to be read then two bytes must be read before any loading WR command can be sent to the same counter.

Read Operation Chart

A1	A0	RD	
0	0	0	Read Counter No. 0
0	1	0	Read Counter No. 1
1	0	0	Read Counter No. 2
1	1	0	Illegal

Reading While Counting

In order for the programmer to read the contents of any counter without effecting or disturbing the counting operation the 8253 has special internal logic that can be accessed using simple WR commands to the MODE register. Basically, when the programmer wishes to read the contents of a selected counter "on the fly" he loads the MODE register with a special code which latches the present count value into a storage register so that its contents contain an accurate, stable quantity. The programmer then issues a normal read command to the selected counter and the contents of the latched register is available.

MODE Register for Latching Count

A0, A1 = 11

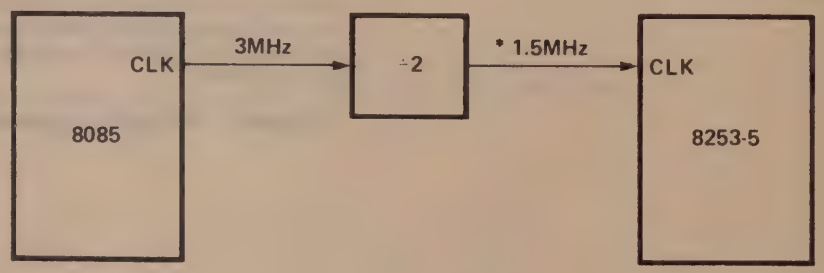
D7	D6	D5	D4	D3	D2	D1	D0
SC1	SC0	0	0	X	X	X	X

SC1,SC0 — specify counter to be latched.

D5,D4 — 00 designates counter latching operation.

X — don't care.

The same limitation applies to this mode of reading the counter as the previous method. That is, it is mandatory to complete the entire read operation as programmed. This command has no effect on the counter's mode.



*If an 8085 clock output is to drive an 8253-5 clock input, it must be reduced to 2 MHz or less.

Figure 10. MCS-85™ Clock Interface*

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias ..... 0° C to 70° C
Storage Temperature ..... −65° C to +150° C
Voltage On Any Pin
With Respect to Ground ..... −0.5 V to +7 V
Power Dissipation ..... 1 Watt

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = 5V ±10%) *

Symbol	Parameter	Min.	Max.	Unit	Test Conditions
V_{IL}	Input Low Voltage	-0.5	0.8	V	
V_{IH}	Input High Voltage	2.2	V_{CC}+.5V	V	
V_{OL}	Output Low Voltage		0.45	V	Note 1
V_{OH}	Output High Voltage	2.4		V	Note 2
I_{IL}	Input Load Current		±10	μA	$V_{IN} = V_{CC}$ to 0V
I_{OFL}	Output Float Leakage		±10	μA	$V_{OUT} = V_{CC}$ to .45V
I_{CC}	V_{CC} Supply Current		140	mA	

CAPACITANCE (T_A = 25°C, V_{CC} = GND = 0V)

Symbol	Parameter	Min.	Typ.	Max.	Unit	Test Conditions
C_{IN}	Input Capacitance			10	pF	fc = 1 MHz
$C_{I/O}$	I/O Capacitance			20	pF	Unmeasured pins returned to V_{SS}

A.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = 5.0V ± 10%, GND = 0V) *

Bus Parameters (Note 3)

READ CYCLE

Symbol	Parameter	8253		8253-5		Unit
		Min.	Max.	Min.	Max.	
t_{AR}	Address Stable Before $\overline{READ}$	50		30		ns
t_{RA}	Address Hold Time for $\overline{READ}$	5		5		ns
t_{RR}	$\overline{READ}$ Pulse Width	400		300		ns
t_{RD}	Data Delay From $\overline{READ}$ [4]		300		200	ns
t_{DF}	$\overline{READ}$ to Data Floating	25	125	25	100	ns
t_{RV}	Recovery Time Between READ and Any Other Control Signal	1		1		μs

A.C. CHARACTERISTICS (Continued)

WRITE CYCLE

Symbol	Parameter	8253 Min.	8253 Max.	8253-5 Min.	8253-5 Max.	Unit
t_{AW}	Address Stable Before $\overline{WRITE}$	50		30		ns
t_{WA}	Address Hold Time for $\overline{WRITE}$	30		30		ns
t_{WW}	$\overline{WRITE}$ Pulse Width	400		300		ns
t_{DW}	Data Set Up Time for $\overline{WRITE}$	300		250		ns
t_{WD}	Data Hold Time for $\overline{WRITE}$	40		30		ns
t_{RV}	Recovery Time Between $\overline{WRITE}$ and Any Other Control Signal	1		1		μs

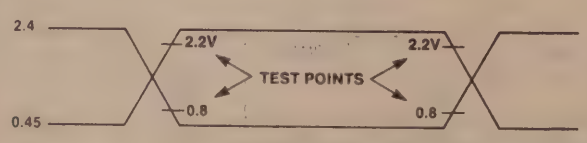
CLOCK AND GATE TIMING

Symbol	Parameter	8253 Min.	8253 Max.	8253-5 Min.	8253-5 Max.	Unit
t_{CLK}	Clock Period	380	dc	380	dc	ns
t_{PWH}	High Pulse Width	230		230		ns
t_{PWL}	Low Pulse Width	150		150		ns
t_{GW}	Gate Width High	150		150		ns
t_{GL}	Gate Width Low	100		100		ns
t_{GS}	Gate Set Up Time to CLK↑	100		100		ns
t_{GH}	Gate Hold Time After CLK↑	50		50		ns
t_{OD}	Output Delay From CLK↓[4]		400		400	ns
t_{ODG}	Output Delay From Gate↓[4]		300		300	ns

NOTES:
1. I_{OL} = 2.2 mA.
2. I_{OH} = −400 μA.
3. AC timings measured at V_{OH} 2.2, V_{OL} = 0.8.
4. C_L = 150pF.

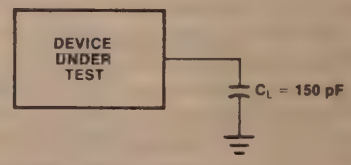
* For Extended Temperature EXPRESS, use M8253 electrical parameters.

A.C. TESTING INPUT, OUTPUT WAVEFORM



A.C. TESTING: INPUTS ARE DRIVEN AT 2.4V FOR A LOGIC "1" AND 0.45V FOR A LOGIC "0." TIMING MEASUREMENTS ARE MADE AT 2.2V FOR A LOGIC "1" AND 0.8V FOR A LOGIC 0.

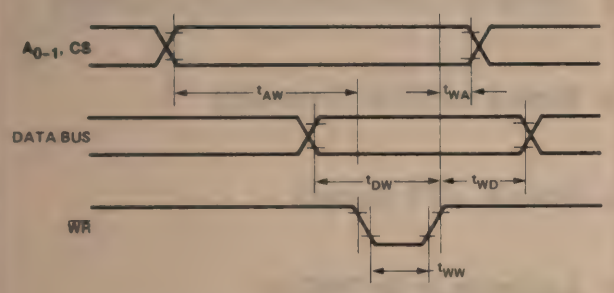
A.C. TESTING LOAD CIRCUIT



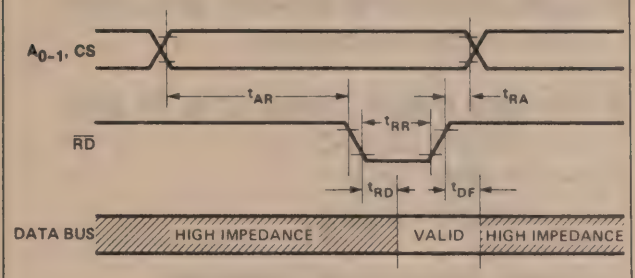
C_L INCLUDES JIG CAPACITANCE

WAVEFORMS

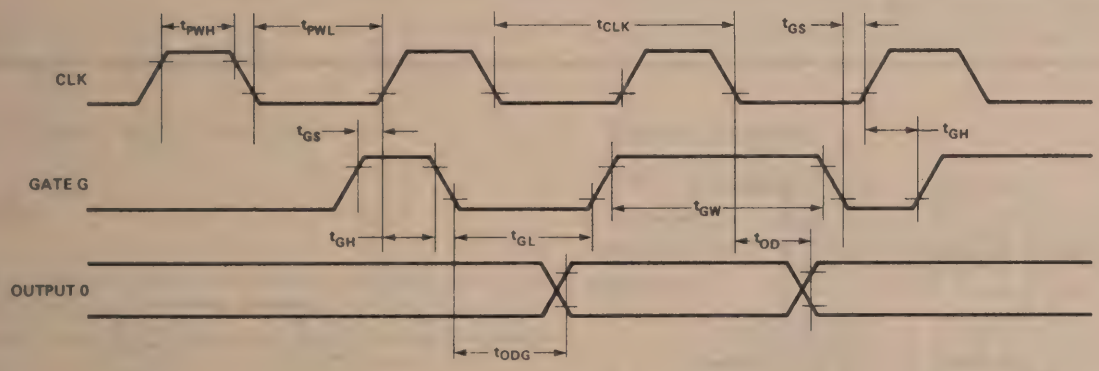
WRITE TIMING



READ TIMING



CLOCK AND GATE TIMING



intel

PRELIMINARY

8254
PROGRAMMABLE INTERVAL TIMER

- Compatible with all Intel and most other microprocessors
- Handles Inputs from DC to 10 MHz
  - 5 MHz 8254-5
  - 8 MHz 8254
  - 10 MHz 8254-2
- Status Read-Back Command
- Six Programmable Counter Modes
- Three Independent 16-bit Counters
- Binary or BCD Counting
- Single +5V Supply
- Available in EXPRESS
  —Standard Temperature Range

The Intel® 8254 is a counter/timer device designed to solve the common timing control problems in microcomputer system design. It provides three independent 16-bit counters, each capable of handling clock inputs up to 10 MHz. All modes are software programmable. The 8254 is a superset of the 8253.

The 8254 uses HMOS technology and comes in a 24-pin plastic or CERDIP package.

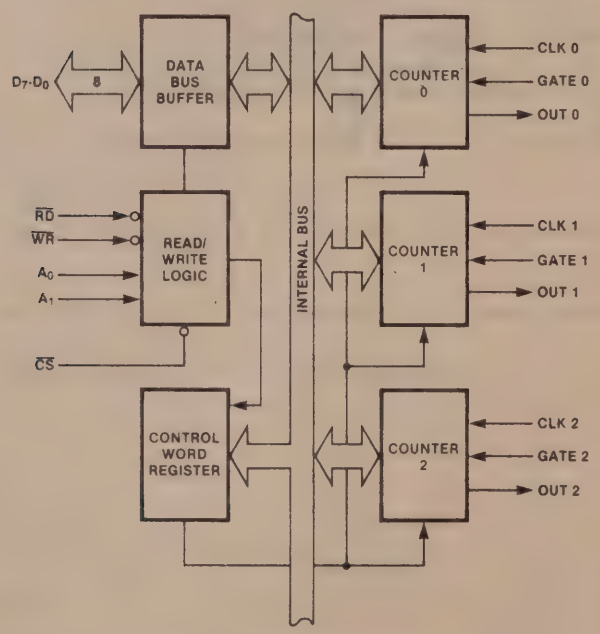


Figure 1. 8254 Block Diagram

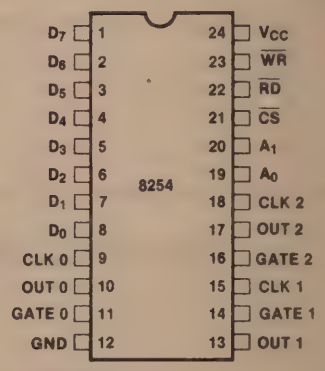


Figure 2. Pin Configuration

Intel Corporation Assumes No Responsibility for the Use of Any Circuitry Other Than Circuitry Embodied in an Intel Product. No Other Circuit Patent Licenses are Implied. Information Contained herein Supersedes Previously Published Specifications On The Devices From Intel.

© INTEL CORPORATION, 1984

January 1985
ORDER NUMBER: 231164-002

Table 1. Pin Description

Symbol	Pin No.	Type	Name and Function
D_7-D_0	1-8	I/O	**Data:** Bi-directional three state data bus lines, connected to system data bus.
CLK 0	9	I	**Clock 0:** Clock Input of Counter 0.
OUT 0	10	O	**Output 0:** Output of Counter 0.
GATE 0	11	I	**Gate 0:** Gate Input of Counter 0.
GND	12		**Ground:** Power supply connection.
V_{CC}	24		**Power:** +5V power supply connection.
$\overline{WR}$	23	I	**Write Control:** This input is low during CPU write operations.
$\overline{RD}$	22	I	**Read Control:** This input is low during CPU read operations.
$\overline{CS}$	21	I	**Chip Select:** A low on this input enables the 8254 to respond to $\overline{RD}$ and $\overline{WR}$ signals. $\overline{RD}$ and $\overline{WR}$ are ignored otherwise.
A_1, A_0	20-19	I	**Address:** Used to select one of the three Counters or the Control Word Register for read or write operations. Normally connected to the system address bus.

A_1	A_0	Selects
0	0	Counter 0
0	1	Counter 1
1	0	Counter 2
1	1	Control Word Register

Symbol	Pin No.	Type	Name and Function
CLK 2	18	I	**Clock 2:** Clock input of Counter 2.
OUT 2	17	O	**Out 2:** Output of Counter 2.
GATE 2	16	I	**Gate 2:** Gate input of Counter 2.
CLK 1	15	I	**Clock 1:** Clock input of Counter 1.
GATE 1	14	I	**Gate 1:** Gate input of Counter 1.
OUT 1	13	O	**Out 1:** Output of Counter 1.

FUNCTIONAL DESCRIPTION

General

The 8254 is a programmable interval timer/counter designed for use with Intel microcomputer systems. It is a general purpose, multi-timing element that can be treated as an array of I/O ports in the system software.

The 8254 solves one of the most common problems in any microcomputer system, the generation of accurate time delays under software control. Instead of setting up timing loops in software, the programmer configures the 8254 to match his requirements and programs one of the counters for the desired delay. After the desired delay, the 8254 will interrupt the CPU. Software overhead is minimal and variable length delays can easily be accommodated.

Some of the other counter/timer functions common to microcomputers which can be implemented with the 8254 are:

- Real time clock
- Event counter
- Digital one-shot
- Programmable rate generator
- Square wave generator
- Binary rate multiplier
- Complex waveform generator
- Complex motor controller

Block Diagram

DATA BUS BUFFER

This 3-state, bi-directional, 8-bit buffer is used to interface the 8254 to the system bus (see Figure 3).

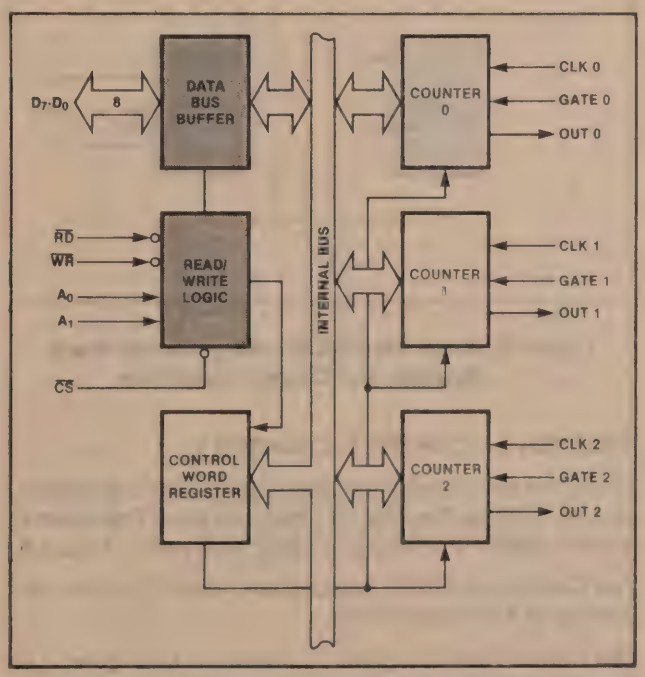


Figure 3. Block Diagram Showing Data Bus Buffer and Read/Write Logic Functions

READ/WRITE LOGIC

The Read/Write Logic accepts inputs from the system bus and generates control signals for the other functional blocks of the 8254. A_1 and A_0 select one of the three counters or the Control Word Register to be read from/written into. A "low" on the $\overline{RD}$ input tells the 8254 that the CPU is reading one of the counters. A "low" on the $\overline{WR}$ input tells the 8254 that the CPU is writing either a Control Word or an initial count. Both $\overline{RD}$ and $\overline{WR}$ are qualified by $\overline{CS}$; $\overline{RD}$ and $\overline{WR}$ are ignored unless the 8254 has been selected by holding $\overline{CS}$ low.

CONTROL WORD REGISTER

The Control Word Register (see Figure 4) is selected by the Read/Write Logic when $A_1,A_0 = 11$. If the CPU then does a write operation to the 8254, the data is stored in the Control Word Register and is interpreted as a Control Word used to define the operation of the Counters.

The Control Word Register can only be written to; status information is available with the Read-Back Command.

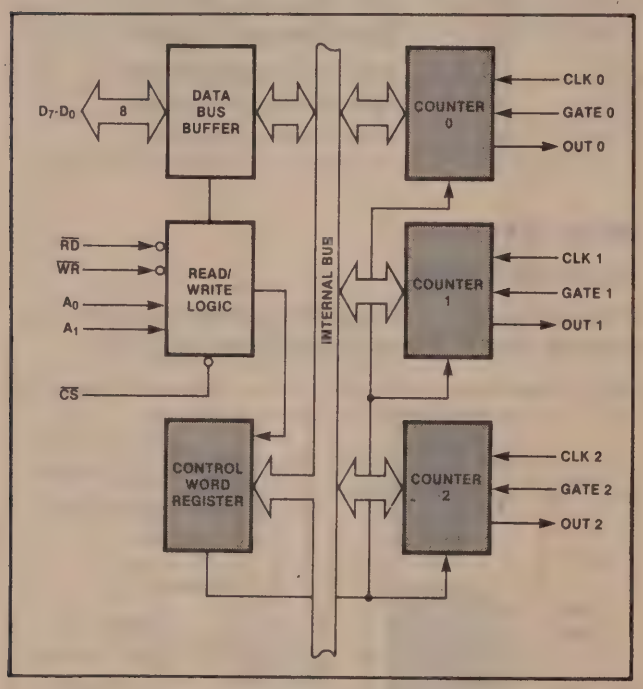


Figure 4. Block Diagram Showing Control Word Register and Counter Functions

COUNTER 0, COUNTER 1, COUNTER 2

These three functional blocks are identical in operation, so only a single Counter will be described. The internal block diagram of a single counter is shown in Figure 5.

The Counters are fully independent. Each Counter may operate in a different Mode.

The Control Word Register is shown in the figure; it is not part of the Counter itself, but its contents determine how the Counter operates.

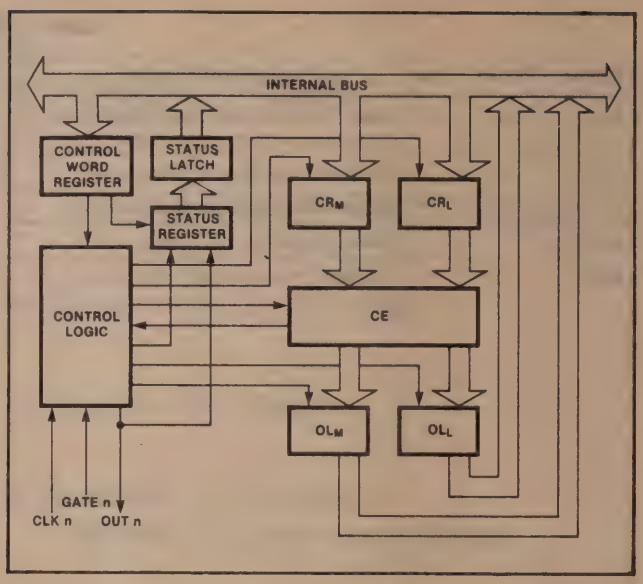


Figure 5. Internal Block Diagram of a Counter

The status register, shown in the Figure, when latched, contains the current contents of the Control Word Register and status of the output and null count flag. (See detailed explanation of the Read-Back command.)

The actual counter is labelled CE (for "Counting Element"). It is a 16-bit presettable synchronous down counter.

OL_M and OL_L are two 8-bit latches. OL stands for "Output Latch"; the subscripts M and L stand for "Most significant byte" and "Least significant byte" respectively. Both are normally referred to as one unit and called just OL. These latches normally "follow" the CE, but if a suitable Counter Latch Command is sent to the 8254, the latches "latch" the present count until read by the CPU and then return to "following" the CE. One latch at a time is enabled by the counter's Control Logic to drive the internal bus. This is how the 16-bit Counter communicates over the 8-bit internal bus. Note that the CE itself cannot be read; whenever you read the count, it is the OL that is being read.

Similarly, there are two 8-bit registers called CR_M and CR_L (for "Count Register"). Both are normally referred to as one unit and called just CR. When a new count is written to the Counter, the count is stored in the CR and later transferred to the CE. The Control Logic allows one register at a time to be loaded from the internal bus. Both bytes are transferred to the CE simultaneously. CR_M and CR_L are cleared when the Counter is programmed. In this way, if the Counter has been programmed for one byte counts (either most significant byte only or least significant byte only) the other byte will be zero. Note that the CE cannot be written into; whenever a count is written, it is written into the CR.

The Control Logic is also shown in the diagram. CLK n, GATE n, and OUT n are all connected to the outside world through the Control Logic.

8254 SYSTEM INTERFACE

The 8254 is a component of the Intel Microcomputer Systems and interfaces in the same manner as all other peripherals of the family. It is treated by the systems software as an array of peripheral I/O ports; three are counters and the fourth is a control register for MODE programming.

Basically, the select inputs A_0, A_1 connect to the A_0, A_1 address bus signals of the CPU. The CS can be derived directly from the address bus using a linear select method. Or it can be connected to the output of a decoder, such as an Intel 8205 for larger systems.

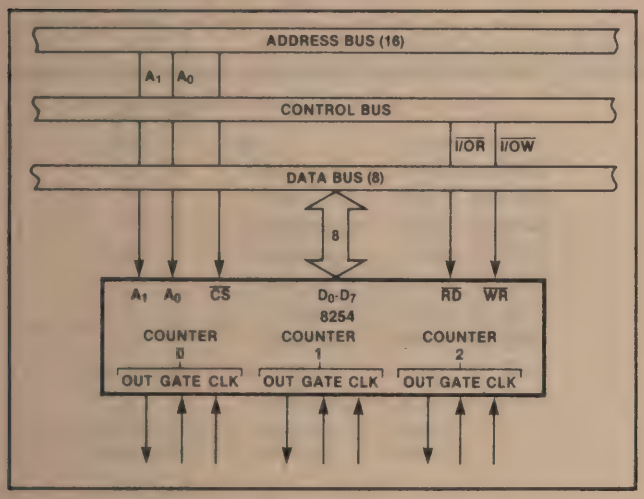


Figure 6. 8254 System Interface

OPERATIONAL DESCRIPTION

General

After power-up, the state of the 8254 is undefined. The Mode, count value, and output of all Counters are undefined.

How each Counter operates is determined when it is programmed. Each Counter must be programmed before it can be used. Unused counters need not be programmed.

Programming the 8254

Counters are programmed by writing a Control Word and then an initial count.

All Control Words are written into the Control Word Register, which is selected when $A_1,A_0 = 11$. The Control Word itself specifies which Counter is being programmed.

By contrast, initial counts are written into the Counters, not the Control Word Register. The A_1,A_0 inputs are used to select the Counter to be written into. The format of the initial count is determined by the Control Word used.

Control Word Format

$A_1,A_0 = 11$ $\overline{CS} = 0$ $\overline{RD} = 1$ $\overline{WR} = 0$

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SC1	SC0	RW1	RW0	M2	M1	M0	BCD

SC — Select Counter:

SC1	SC0	
0	0	Select Counter 0
0	1	Select Counter 1
1	0	Select Counter 2
1	1	Read-Back Command (See Read Operations)

RW — Read/Write:

RW1	RW0	
0	0	Counter Latch Command (see Read Operations)
0	1	Read/Write least significant byte only.
1	0	Read/Write most significant byte only.
1	1	Read/Write least significant byte first, then most significant byte.

M — MODE:

M2	M1	M0	
0	0	0	Mode 0
0	0	1	Mode 1
X	1	0	Mode 2
X	1	1	Mode 3
1	0	0	Mode 4
1	0	1	Mode 5

BCD:

0	Binary Counter 16-bits
1	Binary Coded Decimal (BCD) Counter (4 Decades)

NOTE: DON'T CARE BITS (X) SHOULD BE 0 TO INSURE COMPATIBILITY WITH FUTURE INTEL PRODUCTS.

Figure 7. Control Word Format

Write Operations

The programming procedure for the 8254 is very flexible. Only two conventions need to be remembered:

1) For each Counter, the Control Word must be written before the initial count is written.
2) The initial count must follow the count format specified in the Control Word (least significant byte only, most significant byte only, or least significant byte and then most significant byte).

Since the Control Word Register and the three Counters have separate addresses (selected by the A_1,A_0 inputs), and each Control Word specifies the Counter it applies to (SC0,SC1 bits), no special instruction sequence is required. Any programming sequence that follows the conventions above is acceptable.

A new initial count may be written to a Counter at any time without affecting the Counter's programmed Mode in any way. Counting will be affected as described in the Mode definitions. The new count must follow the programmed count format.

If a Counter is programmed to read/write two-byte counts, the following precaution applies: A program must not transfer control between writing the first and second byte to another routine which also writes into that same Counter. Otherwise, the Counter will be loaded with an incorrect count.

	A_1	A_0
Control Word — Counter 0	1	1
LSB of count — Counter 0	0	0
MSB of count — Counter 0	0	0
Control Word — Counter 1	1	1
LSB of count — Counter 1	0	1
MSB of count — Counter 1	0	1
Control Word — Counter 2	1	1
LSB of count — Counter 2	1	0
MSB of count — Counter 2	1	0

	A_1	A_0
Control Word — Counter 2	1	1
Control Word — Counter 1	1	1
Control Word — Counter 0	1	1
LSB of count — Counter 2	1	0
MSB of count — Counter 2	1	0
LSB of count — Counter 1	0	1
MSB of count — Counter 1	0	1
LSB of count — Counter 0	0	0
MSB of count — Counter 0	0	0

	A_1	A_0
Control Word — Counter 0	1	1
Control Word — Counter 1	1	1
Control Word — Counter 2	1	1
LSB of count — Counter 2	1	0
LSB of count — Counter 1	0	1
LSB of count — Counter 0	0	0
MSB of count — Counter 0	0	0
MSB of count — Counter 1	0	1
MSB of count — Counter 2	1	0

	A_1	A_0
Control Word — Counter 1	1	1
Control Word — Counter 0	1	1
LSB of count — Counter 1	0	1
Control Word — Counter 2	1	1
LSB of count — Counter 0	0	0
MSB of count — Counter 1	0	1
LSB of count — Counter 2	1	0
MSB of count — Counter 0	0	0
MSB of count — Counter 2	1	0

NOTE: IN ALL FOUR EXAMPLES, ALL COUNTERS ARE PROGRAMMED TO READ/WRITE TWO-BYTE COUNTS.
THESE ARE ONLY FOUR OF MANY POSSIBLE PROGRAMMING SEQUENCES.

Figure 8. A Few Possible Programming Sequences

Read Operations

It is often desirable to read the value of a Counter without disturbing the count in progress. This is easily done in the 8254.

There are three possible methods for reading the counters: a simple read operation, the Counter Latch Command, and the Read-Back Command. Each is explained below. The first method is to perform a simple read operation. To read the Counter, which is selected with the A1, A0 inputs, the CLK input of the selected Counter must be inhibited by using either the GATE input or external logic. Otherwise, the count may be in the process of changing when it is read, giving an undefined result.

COUNTER LATCH COMMAND

The second method uses the "Counter Latch Command". Like a Control Word, this command is written to the Control Word Register, which is selected when A_1, A_0= 11. Also like a Control Word, the SC0, SC1 bits select one of the three Counters, but two other bits, D5 and D4, distinguish this command from a Control Word.

A_1,A_0=11; CS=0; RD=1; WR=0

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SC1	SC0	0	0	X	X	X	X

SC1,SC0 — specify counter to be latched

SC1	SC0	Counter
0	0	0
0	1	1
1	0	2
1	1	Read-Back Command

D5,D4 — 00 designates Counter Latch Command

X — don't care

NOTE: DON'T CARE BITS (X) SHOULD BE 0 TO INSURE COMPATIBILITY WITH FUTURE INTEL PRODUCTS.

Figure 9. Counter Latching Command Format

The selected Counter's output latch (OL) latches the count at the time the Counter Latch Command is received. This count is held in the latch until it is read by the CPU (or until the Counter is reprogrammed). The count is then unlatched automatically and the OL returns to "following" the counting element (CE). This allows reading the contents of the Counters "on the fly" without affecting counting in progress. Multiple Counter Latch Commands may be used to latch more than one Counter. Each latched Counter's OL holds its count until it is read. Counter Latch Commands do not affect the programmed Mode of the Counter in any way.

If a Counter is latched and then, some time later, latched again before the count is read, the second Counter Latch Command is ignored. The count read will be the count at the time the first Counter Latch Command was issued.

With either method, the count must be read according to the programmed format; specifically, if the Counter is programmed for two byte counts, two bytes must be read. The two bytes do not have to be read one right after the other; read or write or programming operations of other Counters may be inserted between them.

Another feature of the 8254 is that reads and writes of the same Counter may be interleaved; for example, if the Counter is programmed for two byte counts, the following sequence is valid.

1. Read least significant byte.
2. Write new least significant byte.
3. Read most significant byte.
4. Write new most significant byte.

If a Counter is programmed to read/write two-byte counts, the following precaution applies: A program must not transfer control between reading the first and second byte to another routine which also reads from that same Counter. Otherwise, an incorrect count will be read.

READ-BACK COMMAND

The third method uses the Read-Back Command. This command allows the user to check the count value, programmed Mode, and current states of the OUT pin and Null Count flag of the selected counter(s).

The command is written into the Control Word Register and has the format shown in Figure 10. The command applies to the counters selected by setting their corresponding bits D3,D2,D1=1.

A0, A1=11 $\overline{CS}$=0 $\overline{RD}$=1 $\overline{WR}$=0

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	1	$\overline{COUNT}$	$\overline{STATUS}$	CNT 2	CNT 1	CNT 0	0

D_5: 0 = LATCH COUNT OF SELECTED COUNTER(S)
D_4: 0 = LATCH STATUS OF SELECTED COUNTER(S)
D_3: 1 = SELECT COUNTER 2
D_2: 1 = SELECT COUNTER 1
D_1: 1 = SELECT COUNTER 0
D_0: RESERVED FOR FUTURE EXPANSION; MUST BE 0

Figure 10. Read-Back Command Format

The read-back command may be used to latch multiple counter output latches (OL) by setting the $\overline{COUNT}$ bit D5=0 and selecting the desired counter(s). This single command is functionally equivalent to several counter latch commands, one for each counter latched. Each counter's latched count is held until it is read (or the counter is reprogrammed). That counter is automatically unlatched when read, but other counters remain latched until they are read. If multiple count read-back commands are issued to the same counter without reading the count, all but the first are ignored; i.e., the count which will be read is the count at the time the first read-back command was issued.

The read-back command may also be used to latch status information of selected counter(s) by setting $\overline{STATUS}$ bit D4=0. Status must be latched to be read; status of a counter is accessed by a read from that counter.

The counter status format is shown in Figure 11. Bits D5 through D0 contain the counter's programmed Mode exactly as written in the last Mode Control Word. OUTPUT bit D7 contains the current state of the OUT pin. This allows the user to monitor the counter's output via software, possibly eliminating some hardware from a system.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
OUTPUT	NULL COUNT	RW1	RW0	M2	M1	M0	BCD

D_7 1 = OUT PIN IS 1
0 = OUT PIN IS 0
D_6 1 = NULL COUNT
0 = COUNT AVAILABLE FOR READING
D_5–D_0 COUNTER PROGRAMMED MODE (SEE FIGURE 7)

Figure 11. Status Byte

NULL COUNT bit D6 indicates when the last count written to the counter register (CR) has been loaded into the counting element (CE). The exact time this happens depends on the Mode of the counter and is described in the Mode Definitions, but until the count is loaded into the counting element (CE), it can't be read from the counter. If the count is latched or read before this time, the count value will not reflect the new count just written. The operation of Null Count is shown in Figure 12.

THIS ACTION:	CAUSES:
A. WRITE TO THE CONTROL WORD REGISTER:[1]	NULL COUNT=1
B. WRITE TO THE COUNT REGISTER (CR);[2]	NULL COUNT=1
C. NEW COUNT IS LOADED INTO CE (CR→CE);	NULL COUNT=0

[1] ONLY THE COUNTER SPECIFIED BY THE CONTROL WORD WILL HAVE ITS NULL COUNT SET TO 1. NULL COUNT BITS OF OTHER COUNTERS ARE UNAFFECTED.

[2] IF THE COUNTER IS PROGRAMMED FOR TWO-BYTE COUNTS (LEAST SIGNIFICANT BYTE THEN MOST SIGNIFICANT BYTE) NULL COUNT GOES TO 1 WHEN THE SECOND BYTE IS WRITTEN.

Figure 12. Null Count Operation

If multiple status latch operations of the counter(s) are performed without reading the status, all but the first are ignored; i.e., the status that will be read is the status of the counter at the time the first status read-back command was issued.

Both count and status of the selected counter(s) may be latched simultaneously by setting both $\overline{\text{COUNT}}$ and $\overline{\text{STATUS}}$ bits D5,D4=0. This is functionally the same as issuing two separate read-back commands at once, and the above discussions apply here also. Specifically, if multiple count and/or status read-back commands are issued to the same counter(s) without any intervening reads, all but the first are ignored. This is illustrated in Figure 13.

Command D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Description	Result
1	1	0	0	0	0	1	0	Read back count and status of Counter 0	Count and status latched for Counter 0
1	1	1	0	0	1	0	0	Read back status of Counter 1	Status latched for Counter 1
1	1	1	0	1	1	0	0	Read back status of Counters 2, 1	Status latched for Counter 2, but not Counter 1
1	1	0	1	1	0	0	0	Read back count of Counter 2	Count latched for Counter 2
1	1	0	0	0	1	0	0	Read back count and status of Counter 1	Count latched for Counter 1, but not status
1	1	1	0	0	0	1	0	Read back status of Counter 1	Command ignored, status already latched for Counter 1

Figure 13. Read-Back Command Example

If both count and status of a counter are latched, the first read operation of that counter will return latched status, regardless of which was latched first. The next one or two reads (depending on whether the counter is programmed for one or two type counts) return latched count. Subsequent reads return unlatched count.

$\overline{CS}$	$\overline{RD}$	$\overline{WR}$	A_1	A_0	
0	1	0	0	0	Write into Counter 0
0	1	0	0	1	Write into Counter 1
0	1	0	1	0	Write into Counter 2
0	1	0	1	1	Write Control Word
0	0	1	0	0	Read from Counter 0
0	0	1	0	1	Read from Counter 1
0	0	1	1	0	Read from Counter 2
0	0	1	1	1	No-Operation (3-State)
1	X	X	X	X	No-Operation (3-State)
0	1	1	X	X	No-Operation (3-State)

Figure 14. Read/Write Operations Summary

Mode Definitions

The following are defined for use in describing the operation of the 8254.

CLK pulse: a rising edge, then a falling edge, in that order, of a Counter's CLK input.

trigger: a rising edge of a Counter's GATE input.

Counter loading: the transfer of a count from the CR to the CE (refer to the "Functional Description")

MODE 0: INTERRUPT ON TERMINAL COUNT

Mode 0 is typically used for event counting. After the Control Word is written, OUT is initially low, and will remain low until the Counter reaches zero. OUT then goes high and remains high until a new count or a new Mode 0 Control Word is written into the Counter.

GATE = 1 enables counting; GATE = 0 disables counting. GATE has no effect on OUT.

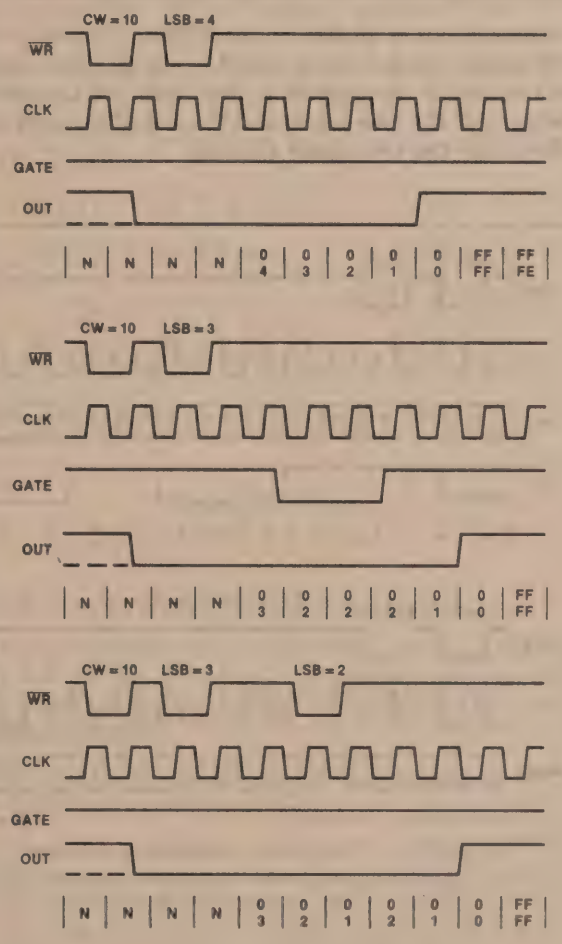
After the Control Word and initial count are written to a Counter, the initial count will be loaded on the next CLK pulse. This CLK pulse does not decrement the count, so for an initial count of N, OUT does not go high until N + 1 CLK pulses after the initial count is written.

If a new count is written to the Counter, it will be loaded on the next CLK pulse and counting will continue from the new count. If a two-byte count is written, the following happens:

1) Writing the first byte disables counting. OUT is set low immediately (no clock pulse required)
2) Writing the second byte allows the new count to be loaded on the next CLK pulse.

This allows the counting sequence to be synchronized by software. Again, OUT does not go high until N + 1 CLK pulses after the new count of N is written.

If an initial count is written while GATE = 0, it will still be loaded on the next CLK pulse. When GATE goes high, OUT will go high N CLK pulses later; no CLK pulse is needed to load the Counter as this has already been done.



NOTE: THE FOLLOWING CONVENTIONS APPLY TO ALL MODE TIMING DIAGRAMS:
1. COUNTERS ARE PROGRAMMED FOR BINARY (NOT BCD) COUNTING AND FOR READING/WRITING LEAST SIGNIFICANT BYTE (LSB) ONLY.
2. THE COUNTER IS ALWAYS SELECTED ($\overline{CS}$ ALWAYS LOW).
3. CW STANDS FOR "CONTROL WORD"; CW = 10 MEANS A CONTROL WORD OF 10, HEX IS WRITTEN TO THE COUNTER.
4. LSB STANDS FOR "LEAST SIGNIFICANT BYTE" OF COUNT.
5. NUMBERS BELOW DIAGRAMS ARE COUNT VALUES.
   THE LOWER NUMBER IS THE LEAST SIGNIFICANT BYTE.
   THE UPPER NUMBER IS THE MOST SIGNIFICANT BYTE. SINCE THE COUNTER IS PROGRAMMED TO READ/WRITE LSB ONLY, THE MOST SIGNIFICANT BYTE CANNOT BE READ.
   N STANDS FOR AN UNDEFINED COUNT.
   VERTICAL LINES SHOW TRANSITIONS BETWEEN COUNT VALUES.

Figure 15. Mode 0

MODE 1: HARDWARE RETRIGGERABLE ONE-SHOT

OUT will be initially high. OUT will go low on the CLK pulse following a trigger to begin the one-shot pulse, and will remain low until the Counter reaches zero. OUT will then go high and remain high until the CLK pulse after the next trigger.

After writing the Control Word and initial count, the Counter is armed. A trigger results in loading the Counter and setting OUT low on the next CLK pulse, thus starting the one-shot pulse. An initial count of N will result in a one-shot pulse N CLK cycles in duration. The one-shot is retriggerable, hence OUT will remain low for N CLK pulses after any trigger. The one-shot pulse can be repeated without rewriting the same count into the counter. GATE has no effect on OUT.

If a new count is written to the Counter during a one-shot pulse, the current one-shot is not affected unless the Counter is retriggered. In that case, the Counter is loaded with the new count and the one-shot pulse continues until the new count expires.

MODE 2: RATE GENERATOR

This Mode functions like a divide-by-N counter. It is typiclaly used to generate a Real Time Clock interrupt. OUT will initially be high. When the initial count has decremented to 1, OUT goes low for one CLK pulse. OUT then goes high again, the Counter reloads the initial count and the process is repeated. Mode 2 is periodic; the same sequence is repeated indefinitely. For an initial count of N, the sequence repeats every N CLK cycles.

GATE = 1 enables counting; GATE = 0 disables counting. If GATE goes low during an output pulse, OUT is set high immediately. A trigger reloads the Counter with the initial count on the next CLK pulse; OUT goes low N CLK pulses after the trigger. Thus the GATE input can be used to synchronize the Counter.

After writing a Control Word and initial count, the Counter will be loaded on the next CLK pulse. OUT goes low N CLK Pulses after the initial count is written. This allows the Counter to be synchronized by software also.

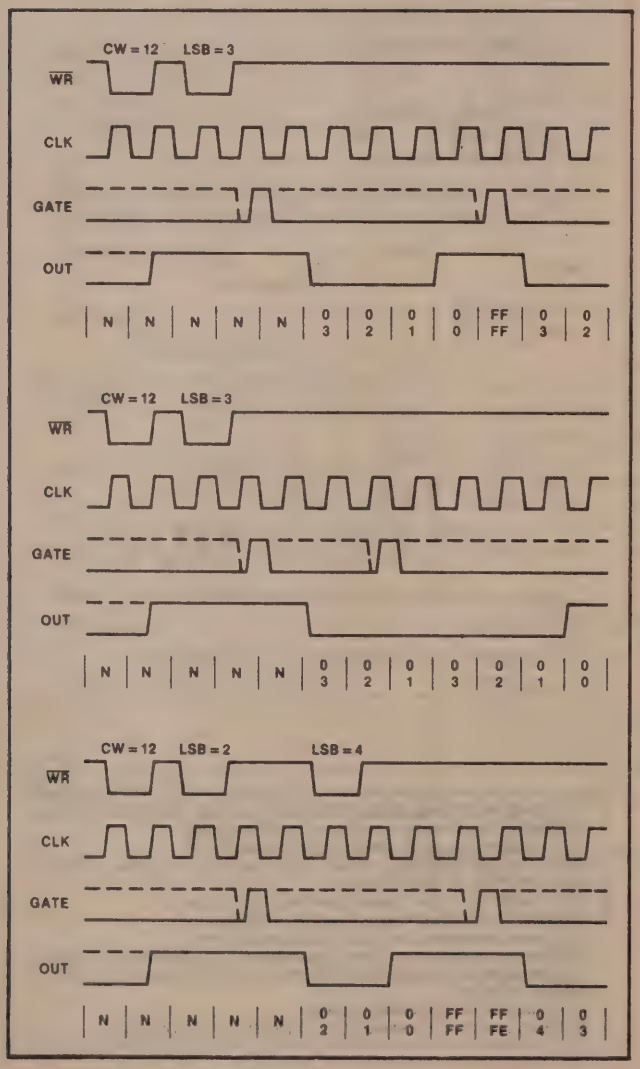
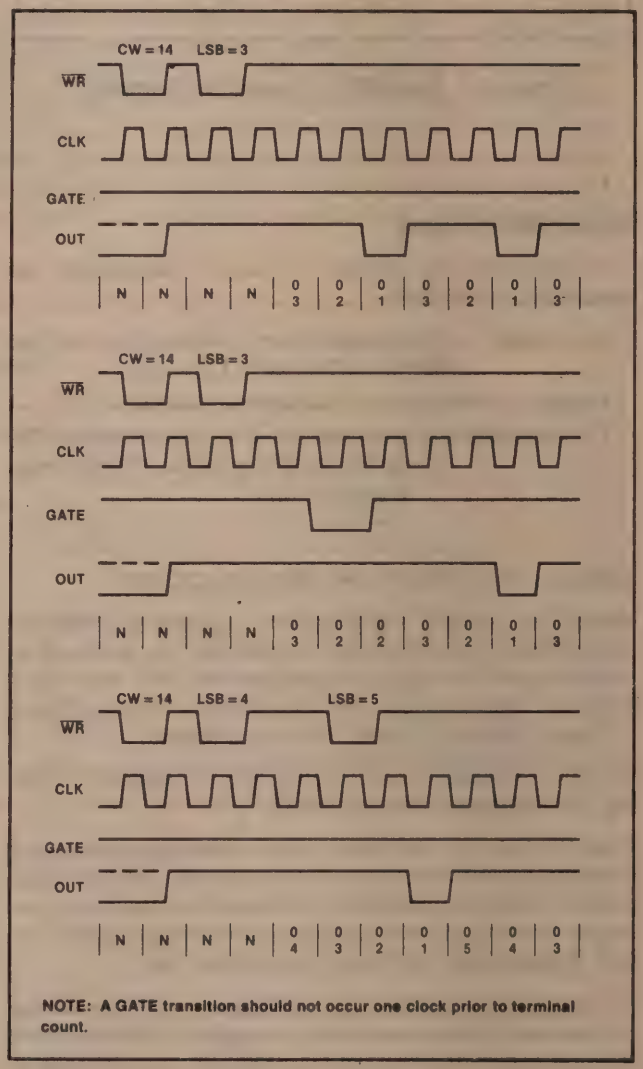


Figure 16. Mode 1



NOTE: A GATE transition should not occur one clock prior to terminal count.

Figure 17. Mode 2

Writing a new count while counting does not affect the current counting sequence. If a trigger is received after writing a new count but before the end of the current period, the Counter will be loaded with the new count on the next CLK pulse and counting will continue from the new count. Otherwise, the new count will be loaded at the end of the current counting cycle. In mode 2, a COUNT of 1 is illegal.

MODE 3: SQUARE WAVE MODE

Mode 3 is typically used for Baud rate generation. Mode 3 is similar to Mode 2 except for the duty cycle of OUT. OUT will initially be high. When half the initial count has expired, OUT goes low for the remainder of the count. Mode 3 is periodic; the sequence above is repeated indefinitely. An initial count of N results in a square wave with a period of N CLK cycles.

GATE = 1 enables counting; GATE = 0 disables counting. If GATE goes low while OUT is low, OUT is set high immediately; no CLK pulse is required. A trigger reloads the Counter with the initial count on the next CLK pulse. Thus the GATE input can be used to synchronize the Counter.

After writing a Control Word and initial count, the Counter will be loaded on the next CLK pulse. This allows the Counter to be synchronized by software also.

Writing a new count while counting does not affect the current counting sequence. If a trigger is received after writing a new count but before the end of the current half-cycle of the square wave, the Counter will be loaded with the new count on the next CLK pulse and counting will continue from the new count. Otherwise, the new count will be loaded at the end of the current half-cycle.

Mode 3 is implemented as follows:

Even counts: OUT is initially high. The initial count is loaded on one CLK pulse and then is decremented by two on succeeding CLK pulses. When the count expires OUT changes value and the Counter is reloaded with the initial count. The above process is repeated indefinitely.

Odd counts: OUT is initially high. The initial count minus one (an even number) is loaded on one CLK pulse and then is decremented by two on succeeding CLK pulses. One CLK pulse *after* the count expires, OUT goes low and the Counter is reloaded with the initial count minus one. Succeeding CLK pulses decrement the count by two. When the count expires, OUT goes high again and the Counter is reloaded with the initial count minus one. The above process is repeated indefinitely. So for odd counts, OUT will be high for (N + 1)/2 counts and low for (N – 1)/2 counts.

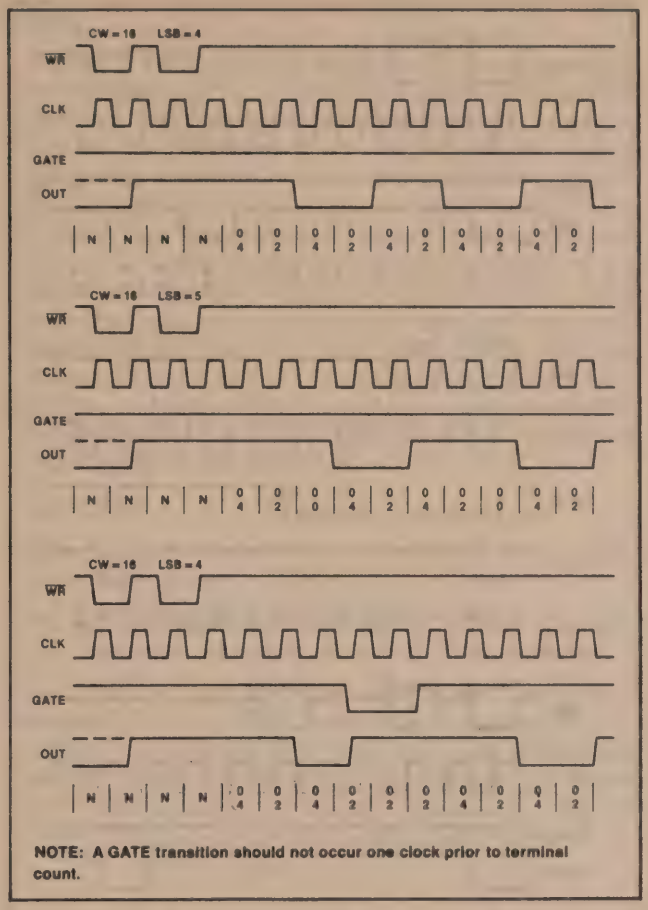


Figure 18. Mode 3

MODE 4: SOFTWARE TRIGGERED STROBE

OUT will be initially high. When the initial count expires, OUT will go low for one CLK pulse and then go high again. The counting sequence is "triggered" by writing the initial count.

GATE = 1 enables counting; GATE = 0 disables counting. GATE has no effect on OUT.

After writing a Control Word and initial count, the Counter will be loaded on the next CLK pulse. This CLK pulse does not decrement the count, so for an initial count of N, OUT does not strobe low until N + 1 CLK pulses after the initial count is written.

If a new count is written during counting, it will be loaded on the next CLK pulse and counting will continue from the new count. If a two-byte count is written, the following happens:

1) Writing the first byte has no effect on counting.
2) Writing the second byte allows the new count to be loaded on the next CLK pulse.

This allows the sequence to be "retriggered" by software. OUT strobes low N + 1 CLK pulses after the new count of N is written.

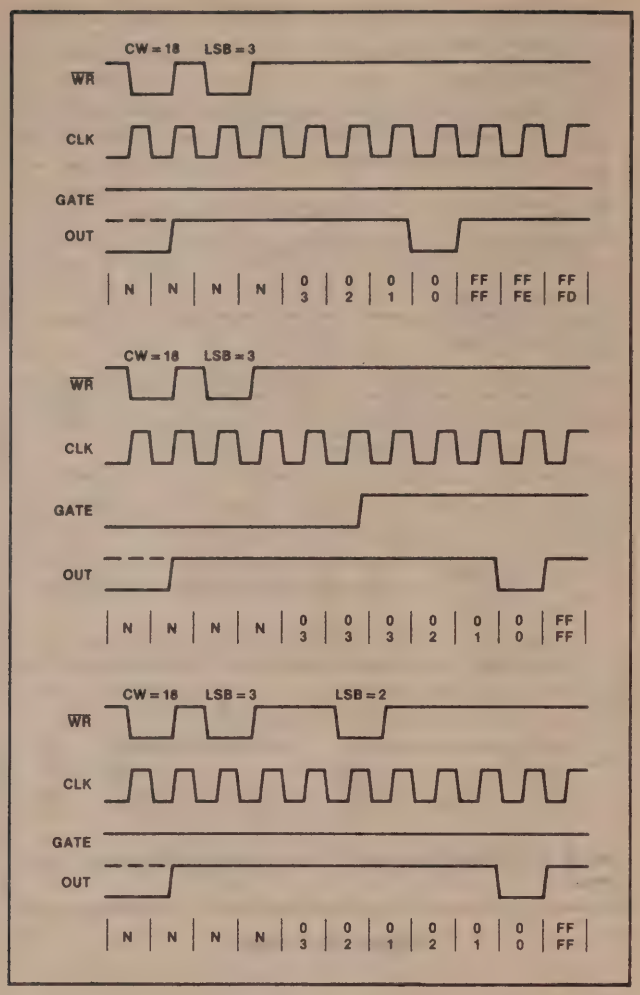


Figure 19. Mode 4

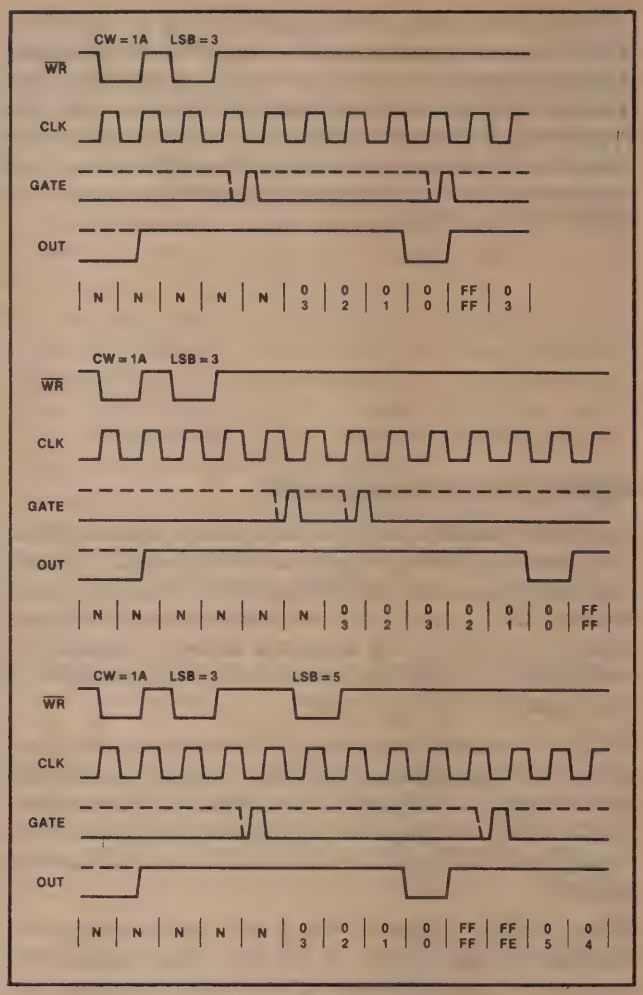


Figure 20. Mode 5

MODE 5: HARDWARE TRIGGERED STROBE (RETRIGGERABLE)

OUT will initially be high. Counting is triggered by a rising edge of GATE. When the initial count has expired, OUT will go low for one CLK pulse and then go high again.

After writing the Control Word and initial count, the counter will not be loaded until the CLK pulse after a trigger. This CLK pulse does not decrement the count, so for an initial count of N, OUT does not strobe low until N + 1 CLK pulses after a trigger.

A trigger results in the Counter being loaded with the initial count on the next CLK pulse. The counting sequence is retriggerable. OUT will not strobe low for N + 1 CLK pulses after any trigger. GATE has no effect on OUT.

If a new count is written during counting, the curent counting sequence will not be affected. If a trigger occurs after the new count is written but before the current count expires, the Counter will be loaded with the new count on the next CLK pulse and counting will continue from there.

Signal Status Modes	Low Or Going Low	Rising	High
0	Disables counting	—	Enables counting
1	—	1) Initiates counting 2) Resets output after next clock	—
2	1) Disables counting 2) Sets output immediately high	Initiates counting	Enables counting
3	1) Disables counting 2) Sets output immediately high	Initiates counting	Enables counting
4	Disables counting	—	Enables counting
5	—	Initiates counting	—

Figure 21. Gate Pin Operations Summary

Mode	Min Count	Max Count
0	1	0
1	1	0
2	2	0
3	2	0
4	1	0
5	1	0

NOTE: 0 IS EQUIVALENT TO 2^{16} FOR BINARY COUNTING AND 10^4 FOR BCD COUNTING.

Figure 22. Minimum and Maximum Initial Counts

Operation Common to All Modes

PROGRAMMING

When a Control Word is written to a Counter, all Control Logic is immediately reset and OUT goes to a known initial state; no CLK pulses are required for this.

GATE

The GATE input is always sampled on the rising edge of CLK. In Modes 0, 2, 3, and 4 the GATE input is level sensitive, and the logic level is sampled on the rising edge of CLK. In Modes 1, 2, 3, and 5 the GATE input is rising-edge sensitive. In these Modes, a rising edge of GATE (trigger) sets an edge-sensitive flip-flop in the Counter. This flip-flop is then sampled on the next rising edge of CLK; the flip-flop is reset immediately after it is sampled. In this way, a trigger will be detected no matter when it occurs—a high logic level does not have to be maintained until the next rising edge of CLK. Note that in Modes 2 and 3, the GATE input is both edge- and level-sensitive. In Modes 2 and 3, if a CLK source other than the system clock is used, GATE should be pulsed immediately following $\overline{WR}$ of a new count value.

COUNTER

New counts are loaded and Counters are decremented on the falling edge of CLK.

The largest possible initial count is 0; this is equivalent to 2^{16} for binary counting and 10^4 for BCD counting.

The Counter does not stop when it reaches zero. In Modes 0, 1, 4, and 5 the Counter "wraps around" to the highest count, either FFFF hex for binary counting or 9999 for BCD counting, and continues counting. Modes 2 and 3 are periodic; the Counter reloads itself with the initial count and continues counting from there.

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias . . . . . 0°C to 70°C

Storage Temperature . . . . . −65°C to +150°C

Voltage on Any Pin with Respect to Ground . . . . . −0.5V to +7V

Power Dissipation . . . . . 1 Watt

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to Absolute Maximum Rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS ($T_A = 0°C$ to $70°C$; $V_{CC} = 5V \pm 10\%$)

Symbol	Parameter	Min.	Max.	Units	Test Conditions
V_{IL}	Input Low Voltage	−0.5	0.8	V	
V_{IH}	Input High Voltage	2.0	$V_{CC} + 0.5V$	V	
V_{OL}	Output Low Voltage		0.45	V	$I_{OL} = 2.0$ mA
V_{OH}	Output High Voltage	2.4		V	$I_{OH} = -400\mu A$
I_{IL}	Input Load Current		±10	μA	$V_{IN} = V_{CC}$ to 0V
I_{OFL}	Output Float Leakage		±10	μA	$V_{OUT} = V_{CC}$ to 0.45V
I_{CC}	V_{CC} Supply Current		170	mA	

CAPACITANCE ($T_A = 25°C$, $V_{CC} = GND = 0V$)

Symbol	Parameter	Min.	Max.	Units	Test Conditions
C_{IN}	Input Capacitance		10	pF	$f_c = 1$ MHz
$C_{I/O}$	I/O Capacitance		20	pF	Unmeasured pins returned to V_{SS}

A.C. CHARACTERISTICS ($T_A = 0°C$ to $70°C$, $V_{CC} = 5V \pm 10\%$, GND = 0V)

Bus Parameters (Note 1)

READ CYCLE

Symbol	Parameter	8254-5		8254		8254-2		Unit
		Min.	Max.	Min.	Max.	Min.	Max.	
t_{AR}	Address Stable Before $\overline{RD}$↓	45		45		30		ns
t_{SR}	$\overline{CS}$ Stable Before $\overline{RD}$↓	0		0		0		ns
t_{RA}	Address Hold Time After $\overline{RD}$↑	0		0		0		ns
t_{RR}	$\overline{RD}$ Pulse Width	150		150		95		ns
t_{RD}	Data Delay from $\overline{RD}$↓		120		120		85	ns
t_{AD}	Data Delay from Address		220		220		185	ns
t_{DF}	$\overline{RD}$↑ to Data Floating	5	90	5	90	5	65	ns
t_{RV}	Command Recovery Time	200		200		165		ns

Note 1: AC timings measured at $V_{OH} = 2.0V$, $V_{OL} = 0.8V$.

A.C. CHARACTERISTICS (Continued)

WRITE CYCLE

Symbol	Parameter	8254-5		8254		8254-2		Unit
		Min.	Max.	Min.	Max.	Min.	Max.	
t_{AW}	Address Stable Before $\overline{WR}$↓	0		0		0		ns
t_{SW}	$\overline{CS}$ Stable Before $\overline{WR}$↓	0		0		0		ns
t_{WA}	Address Hold Time After $\overline{WR}$↑	0		0		0		ns
t_{WW}	$\overline{WR}$ Pulse Width	150		150		95		ns
t_{DW}	Data Setup Time Before $\overline{WR}$↑	120		120		95		ns
t_{WD}	Data Hold time After $\overline{WR}$↑	0		0		0		ns
t_{RV}	Command Recovery Time	200		200		165		ns

CLOCK AND GATE

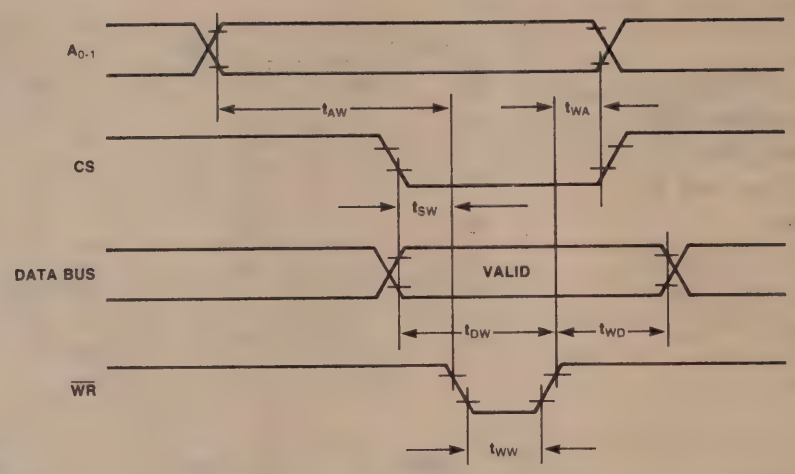
Symbol	Parameter	8254-5		8254		8254-2		Unit
		Min.	Max.	Min.	Max.	Min.	Max.	
t_{CLK}	Clock Period	200	DC	125	DC	100	DC	ns
t_{PWH}	High Pulse Width	60[3]		60[3]		30[3]		ns
t_{PWL}	Low Pulse Width	60[3]		60[3]		50[3]		ns
t_R	Clock Rise Time		25		25		25	ns
t_F	Clock Fall Time		25		25		25	ns
t_{GW}	Gate Width High	50		50		50		ns
t_{GL}	Gate Width Low	50		50		50		ns
t_{GS}	Gate Setup Time to CLK↑	50		50		40		ns
t_{GH}	Gate Setup Time After CLK↑	50[2]		50[2]		50[2]		ns
t_{OD}	Output Delay from CLK↓		150		150		100	ns
t_{ODG}	Output Delay from Gate↓		120		120		100	ns
t_{WC}	CLK Delay for Loading↓	0	55	0	55	0	55	ns
t_{WG}	Gate Delay for Sampling	-5	50	-5	50	-5	40	ns
t_{WO}	OUT Delay from Mode Write		260		260		240	ns
t_{CL}	CLK Set Up for Count Latch	-40	45	-40	45	-40	40	ns

Note 2: In Modes 1 and 5 triggers are sampled on each rising clock edge. A second trigger within 120 ns (70 ns for the 8254-2) of the rising clock edge may not be detected.

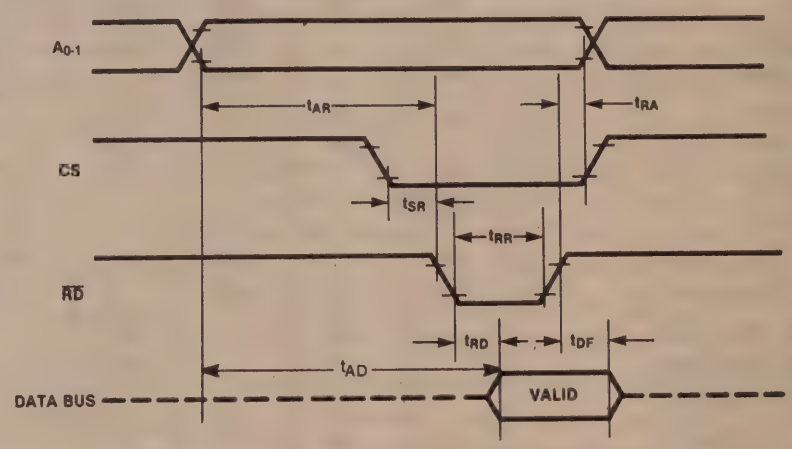
Note 3: Low-going glitches that violate t_{PWH}, t_{PWL} may cause errors requiring counter reprogramming.

WAVEFORMS

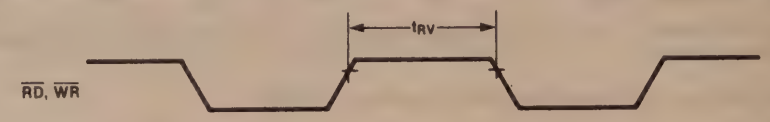
WRITE



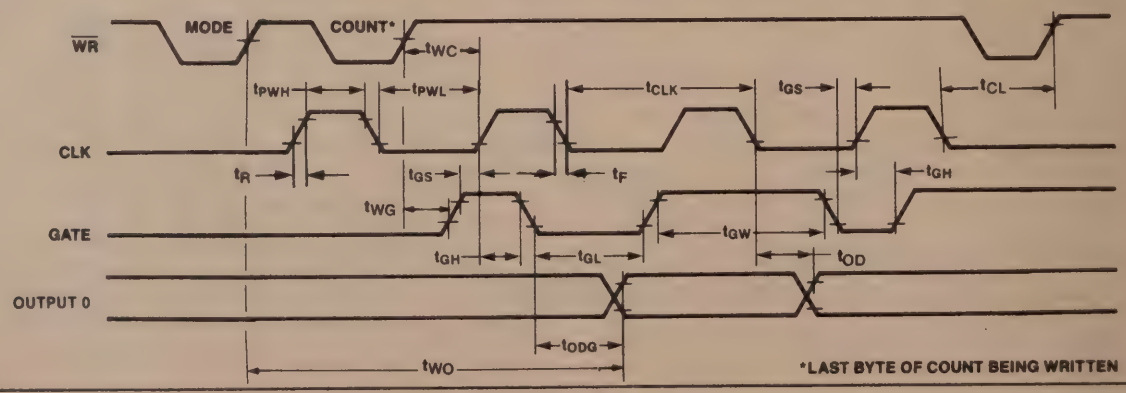
READ



RECOVERY

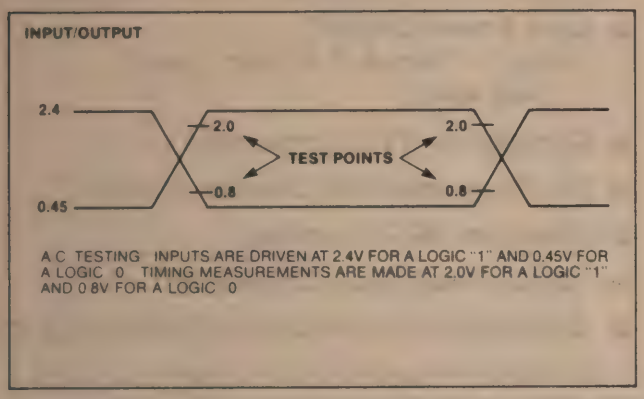


CLOCK AND GATE



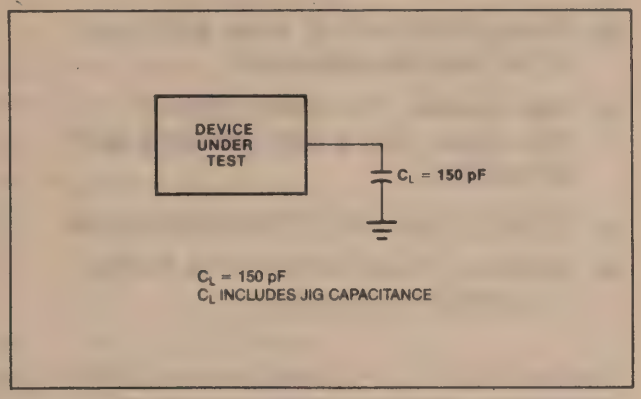
231164-002

A.C. TESTING INPUT, OUTPUT WAVEFORM



A C TESTING INPUTS ARE DRIVEN AT 2.4V FOR A LOGIC "1" AND 0.45V FOR A LOGIC 0 TIMING MEASUREMENTS ARE MADE AT 2.0V FOR A LOGIC "1" AND 0 8V FOR A LOGIC 0

A.C. TESTING LOAD CIRCUIT



C_L = 150 pF
C_L INCLUDES JIG CAPACITANCE

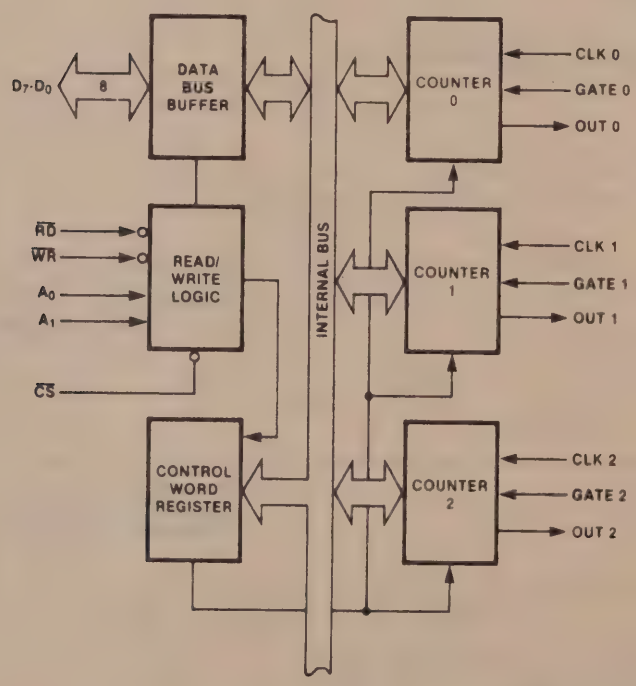
82C54
CHMOS PROGRAMMABLE INTERVAL TIMER

- **Compatible with all Intel and most other microprocessors**
- **High Speed, "Zero Wait State" Operation with 8 MHz 8086/88 and 80186/188**
- **Three independent 16-bit counters**
- **Handles Inputs from DC to 8 MHz — 10 MHz for 82C54-2**
- **Low Power CHMOS — I_{CC} = 10 mA @ 8 MHz Count frequency**
- **Completely TTL Compatible**
- **Six Programmable Counter Modes**
- **Binary or BCD counting**
- **Status Read Back Command**
- **Available in 24-Pin DIP and 28-Pin PLCC**

The Intel 82C54 is a high-performance, CHMOS version of the industry standard 8254 counter/timer which is designed to solve the timing control problems common in microcomputer system design. It provides three independent 16-bit counters, each capable of handling clock inputs up to 10 MHz. All modes are software programmable. The 82C54 is pin compatible with the HMOS 8254, and is a superset of the 8253.

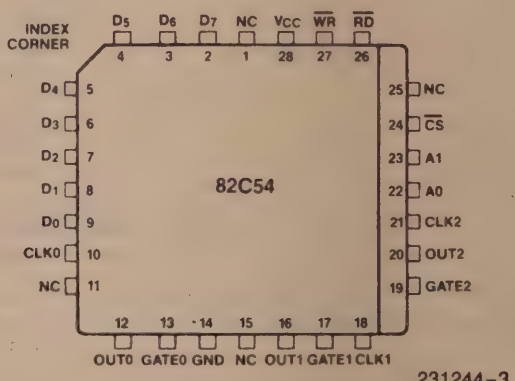
Six programmable timer modes allow the 82C54 to be used as an event counter, elapsed time indicator, programmable one-shot, and in many other applications.

The 82C54 is fabricated on Intel's advanced CHMOS III technology which provides low power consumption with performance equal to or greater than the equivalent HMOS product. The 82C54 is available in 24-pin DIP and 28-pin plastic leaded chip carrier (PLCC) packages.



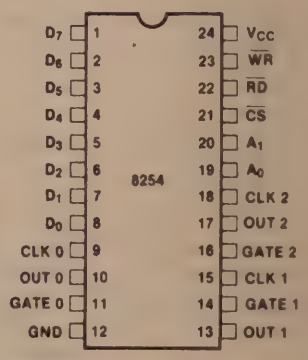
231244-1

Figure 1. 82C54 Block Diagram



231244-3

PLASTIC LEADED CHIP CARRIER



231244-2

Diagrams are for pin reference only.
Package sizes are not to scale.

Figure 2. 82C54 Pinout

Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied. Information contained herein supersedes previously published specifications on these devices from Intel. **November 1985**
© Intel Corporation, 1985

Order Number: 231244-002

Table 1. Pin Description

Symbol	Pin Number DIP	Pin Number PLCC	Type	Function
D_7-D_0	1-8	2-9	I/O	Data: Bidirectional tri-state data bus lines, connected to system data bus.
CLK 0	9	10	I	Clock 0: Clock input of Counter 0.
OUT 0	10	12	O	Output 0: Output of Counter 0.
GATE 0	11	13	I	Gate 0: Gate input of Counter 0.
GND	12	14		Ground: Power supply connection.
OUT 1	13	16	O	Out 1: Output of Counter 1.
GATE 1	14	17	I	Gate 1: Gate input of Counter 1.
CLK 1	15	18	I	Clock 1: Clock input of Counter 1.
GATE 2	16	19	I	Gate 2: Gate input of Counter 2.
OUT 2	17	20	O	Out 2: Output of Counter 2.
CLK 2	18	21	I	Clock 2: Clock input of Counter 2.
A_1, A_0	20-19	23-22	I	Address: Used to select one of the three Counters or the Control Word Register for read or write operations. Normally connected to the system address bus.
				A_1 / A_0 / Selects: 0 0 Counter 0; 0 1 Counter 1; 1 0 Counter 2; 1 1 Control Word Register
$\overline{CS}$	21	24	I	Chip Select: A low on this input enables the 82C54 to respond to $\overline{RD}$ and $\overline{WR}$ signals. $\overline{RD}$ and $\overline{WR}$ are ignored otherwise.
$\overline{RD}$	22	26	I	Read Control: This input is low during CPU read operations.
$\overline{WR}$	23	27	I	Write Control: This input is low during CPU write operations.
V_{CC}	24	28		Power: +5V power supply connection.
NC		1, 11, 15, 25		No Connect

FUNCTIONAL DESCRIPTION

General

The 82C54 is a programmable interval timer/counter designed for use with Intel microcomputer systems. It is a general purpose, multi-timing element that can be treated as an array of I/O ports in the system software.

The 82C54 solves one of the most common problems in any microcomputer system, the generation of accurate time delays under software control. Instead of setting up timing loops in software, the programmer configures the 82C54 to match his requirements and programs one of the counters for the desired delay. After the desired delay, the 82C54 will interrupt the CPU. Software overhead is minimal and variable length delays can easily be accommodated.

Some of the other counter/timer functions common to microcomputers which can be implemented with the 82C54 are:

- Real time clock
- Even counter
- Digital one-shot
- Programmable rate generator
- Square wave generator
- Binary rate multiplier
- Complex waveform generator
- Complex motor controller

Block Diagram

DATA BUS BUFFER

This 3-state, bi-directional, 8-bit buffer is used to interface the 82C54 to the system bus (see Figure 3).

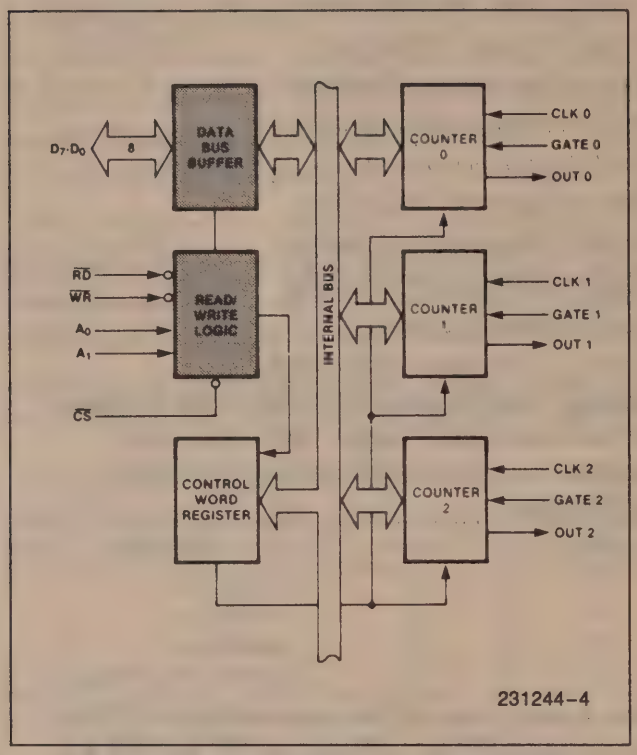


Figure 3. Block Diagram Showing Data Bus Buffer and Read/Write Logic Functions

READ/WRITE LOGIC

The Read/Write Logic accepts inputs from the system bus and generates control signals for the other functional blocks of the 82C54. A_1 and A_0 select one of the three counters or the Control Word Register to be read from/written into. A "low" on the $\overline{RD}$ input tells the 82C54 that the CPU is reading one of the counters. A "low" on the $\overline{WR}$ input tells the 82C54 that the CPU is writing either a Control Word or an initial count. Both $\overline{RD}$ and $\overline{WR}$ are qualified by $\overline{CS}$; $\overline{RD}$ and $\overline{WR}$ are ignored unless the 82C54 has been selected by holding $\overline{CS}$ low.

CONTROL WORD REGISTER

The Control Word Register (see Figure 4) is selected by the Read/Write Logic when A_1, A_0 = 11. If the CPU then does a write operation to the 82C54, the data is stored in the Control Word Register and is interpreted as a Control Word used to define the operation of the Counters.

The Control Word Register can only be written to; status information is available with the Read-Back Command.

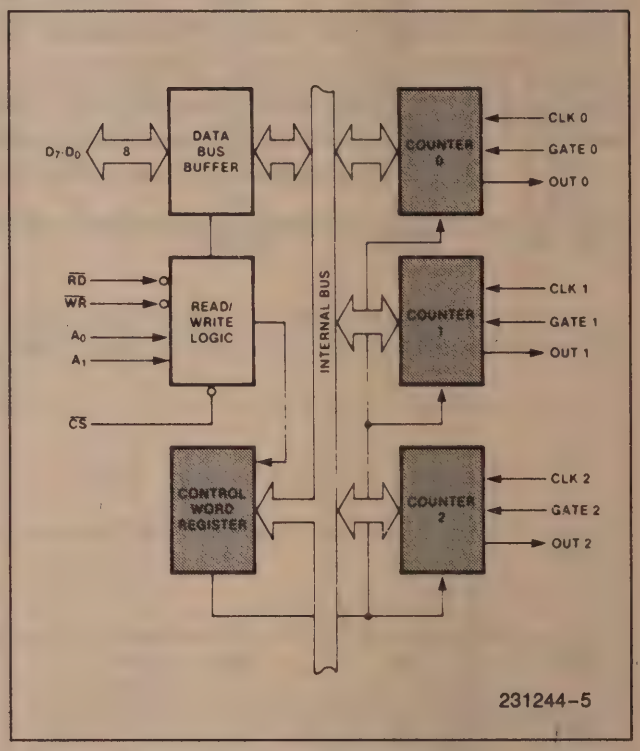


Figure 4. Block Diagram Showing Control Word Register and Counter Functions

COUNTER 0, COUNTER 1, COUNTER 2

These three functional blocks are identical in operation, so only a single Counter will be described. The internal block diagram of a single counter is shown in Figure 5.

The Counters are fully independent. Each Counter may operate in a different Mode.

The Control Word Register is shown in the figure; it is not part of the Counter itself, but its contents determine how the Counter operates.

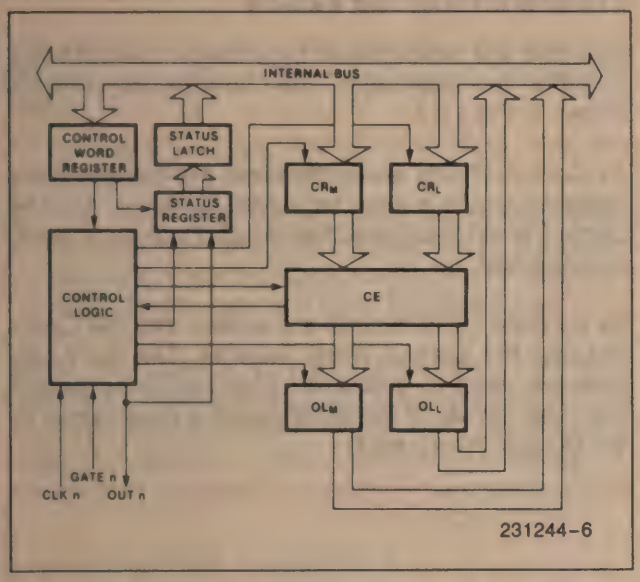


Figure 5. Internal Block Diagram of a Counter

The status register, shown in the Figure, when latched, contains the current contents of the Control Word Register and status of the output and null count flag. (See detailed explanation of the Read-Back command.)

The actual counter is labelled CE (for "Counting Element"). It is a 16-bit presettable synchronous down counter.

OL_M and OL_L are two 8-bit latches. OL stands for "Output Latch"; the subscripts M and L stand for "Most significant byte" and "Least significant byte" respectively. Both are normally referred to as one unit and called just OL. These latches normally "follow" the CE, but if a suitable Counter Latch Command is sent to the 82C54, the latches "latch" the present count until read by the CPU and then return to "following" the CE. One latch at a time is enabled by the counter's Control Logic to drive the internal bus. This is how the 16-bit Counter communicates over the 8-bit internal bus. Note that the CE itself cannot be read; whenever you read the count, it is the OL that is being read.

Similarly, there are two 8-bit registers called CR_M and CR_L (for "Count Register"). Both are normally referred to as one unit and called just CR. When a new count is written to the Counter, the count is stored in the CR and later transferred to the CE. The Control Logic allows one register at a time to be loaded from the internal bus. Both bytes are transferred to the CE simultaneously. CR_M and CR_L are cleared when the Counter is programmed. In this way, if the Counter has been programmed for one byte counts (either most significant byte only or least significant byte only) the other byte will be zero. Note that the CE cannot be written into; whenever a count is written, it is written into the CR.

The Control Logic is also shown in the diagram. CLK n, GATE n, and OUT n are all connected to the outside world through the Control Logic.

82C54 SYSTEM INTERFACE

The 82C54 is treated by the systems software as an array of peripheral I/O ports; three are counters and the fourth is a control register for MODE programming.

Basically, the select inputs A_0, A_1 connect to the A_0, A_1 address bus signals of the CPU. The $\overline{CS}$ can be derived directly from the address bus using a linear select method. Or it can be connected to the output of a decoder, such as an Intel 8205 for larger systems.

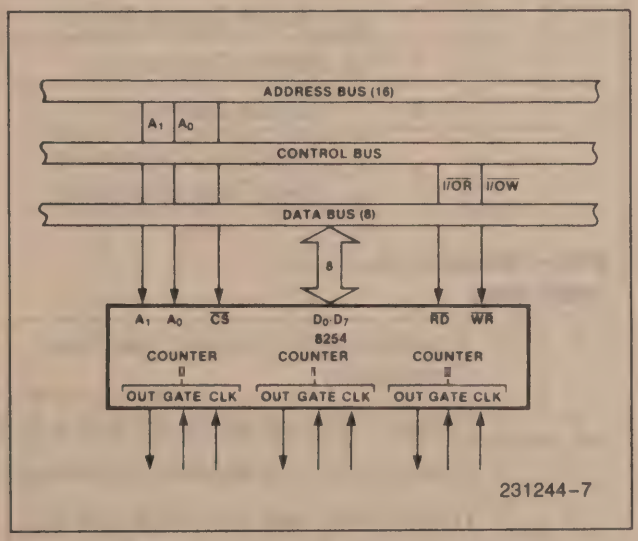


Figure 6. 82C54 System Interface

OPERATIONAL DESCRIPTION

General

After power-up, the state of the 82C54 is undefined. The Mode, count value, and output of all Counters are undefined.

How each Counter operates is determined when it is programmed. Each Counter must be programmed before it can be used. Unused counters need not be programmed.

Programming the 82C54

Counters are programmed by writing a Control Word and then an initial count. The control word format is shown in Figure 7.

All Control Words are written into the Control Word Register, which is selected when A_1, A_0 = 11. The Control Word itself specifies which Counter is being programmed.

By contrast, initial counts are written into the Counters, not the Control Word Register. The A_1, A_0 inputs are used to select the Counter to be written into. The format of the initial count is determined by the Control Word used.

Control Word Format

$A_1, A_0 = 11$ $\overline{CS} = 0$ $\overline{RD} = 1$ $\overline{WR} = 0$

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SC1	SC0	RW1	RW0	M2	M1	M0	BCD

SC — Select Counter:

SC1	SC0	
0	0	Select Counter 0
0	1	Select Counter 1
1	0	Select Counter 2
1	1	Read-Back Command (See Read Operations)

RW — Read/Write:

RW1	RW0	
0	0	Counter Latch Command (see Read Operations)
0	1	Read/Write least significant byte only.
1	0	Read/Write most significant byte only.
1	1	Read/Write least significant byte first, then most significant byte.

M — MODE:

M2	M1	M0	
0	0	0	Mode 0
0	0	1	Mode 1
X	1	0	Mode 2
X	1	1	Mode 3
1	0	0	Mode 4
1	0	1	Mode 5

BCD:

0	Binary Counter 16-bits
1	Binary Coded Decimal (BCD) Counter (4 Decades)

NOTE: Don't care bits (X) should be 0 to insure compatibility with future Intel products.

Figure 7. Control Word Format

Write Operations

The programming procedure for the 82C54 is very flexible. Only two conventions need to be remembered:

1) For each Counter, the Control Word must be written before the initial count is written.
2) The initial count must follow the count format specified in the Control Word (least significant byte only, most significant byte only, or least significant byte and then most significant byte).

Since the Control Word Register and the three Counters have separate addresses (selected by the A_1, A_0 inputs), and each Control Word specifies the Counter it applies to (SC0, SC1 bits), no special instruction sequence is required. Any programming sequence that follows the conventions above is acceptable.

A new initial count may be written to a Counter at any time without affecting the Counter's programmed Mode in any way. Counting will be affected as described in the Mode definitions. The new count must follow the programmed count format.

If a Counter is programmed to read/write two-byte counts, the following precaution applies: A program must not transfer control between writing the first and second byte to another routine which also writes into that same Counter. Otherwise, the Counter will be loaded with an incorrect count.

		A_1	A_0
Control Word —	Counter 0	1	1
LSB of count —	Counter 0	0	0
MSB of count —	Counter 0	0	0
Control Word —	Counter 1	1	1
LSB of count —	Counter 1	0	1
MSB of count —	Counter 1	0	1
Control Word —	Counter 2	1	1
LSB of count —	Counter 2	1	0
MSB of count —	Counter 2	1	0

		A_1	A_0
Control Word —	Counter 2	1	1
Control Word —	Counter 1	1	1
Control Word —	Counter 0	1	1
LSB of count —	Counter 2	1	0
MSB of count —	Counter 2	1	0
LSB of count —	Counter 1	0	1
MSB of count —	Counter 1	0	1
LSB of count —	Counter 0	0	0
MSB of count —	Counter 0	0	0

		A_1	A_0
Control Word —	Counter 0	1	1
Counter Word —	Counter 1	1	1
Control Word —	Counter 2	1	1
LSB of count —	Counter 2	1	0
LSB of count —	Counter 1	0	1
LSB of count —	Counter 0	0	0
MSB of count —	Counter 0	0	0
MSB of count —	Counter 1	0	1
MSB of count —	Counter 2	1	0

		A_1	A_0
Control Word —	Counter 1	1	1
Control Word —	Counter 0	1	1
LSB of count —	Counter 1	0	1
Control Word —	Counter 2	1	1
LSB of count —	Counter 0	0	0
MSB of count —	Counter 1	0	1
LSB of count —	Counter 2	1	0
MSB of count —	Counter 0	0	0
MSB of count —	Counter 2	1	0

NOTE:
In all four examples, all counters are programmed to read/write two-byte counts.
These are only four of many possible programming sequences.

Figure 8. A Few Possible Programming Sequences

Read Operations

It is often desirable to read the value of a Counter without disturbing the count in progress. This is easily done in the 82C54.

There are three possible methods for reading the counters: a simple read operation, the Counter Latch Command, and the Read-Back Command. Each is explained below. The first method is to perform a simple read operation. To read the Counter, which is selected with the A1, A0 inputs, the CLK input of the selected Counter must be inhibited by using either the GATE input or external logic. Otherwise, the count may be in the process of changing when it is read, giving an undefined result.

COUNTER LATCH COMMAND

The second method uses the "Counter Latch Command". Like a Control Word, this command is written to the Control Word Register, which is selected when A_1, A_0 = 11. Also like a Control Word, the SC0, SC1 bits select one of the three Counters, but two other bits, D5 and D4, distinguish this command from a Control Word.

A_1, A_0 = 11; $\overline{CS}$ = 0; $\overline{RD}$ = 1; $\overline{WR}$ = 0

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SC1	SC0	0	0	X	X	X	X

SC1, SC0 - specify counter to be latched

SC1	SC0	Counter
0	0	0
0	1	1
1	0	2
1	1	Read-Back Command

D5,D4 - 00 designates Counter Latch Command

X - don't care

NOTE:
Don't care bits (X) should be 0 to insure compatibility with future Intel products.

Figure 9. Counter Latching Command Format

The selected Counter's output latch (OL) latches the count at the time the Counter Latch Command is received. This count is held in the latch until it is read by the CPU (or until the Counter is reprogrammed). The count is then unlatched automatically and the OL returns to "following" the counting element (CE). This allows reading the contents of the Counters "on the fly" without affecting counting in progress. Multiple Counter Latch Commands may be used to latch more than one Counter. Each latched Counter's OL holds its count until it is read. Counter Latch Commands do not affect the programmed Mode of the Counter in any way.

If a Counter is latched and then, some time later, latched again before the count is read, the second Counter Latch Command is ignored. The count read will be the count at the time the first Counter Latch Command was issued.

With either method, the count must be read according to the programmed format; specifically, if the Counter is programmed for two byte counts, two bytes must be read. The two bytes do not have to be read one right after the other; read or write or programming operations of other Counters may be inserted between them.

Another feature of the 82C54 is that reads and writes of the same Counter may be interleaved; for example, if the Counter is programmed for two byte counts, the following sequence is valid.

1. Read least significant byte.
2. Write new least significant byte.
3. Read most significant byte.
4. Write new most significant byte.

If a Counter is programmed to read/write two-byte counts, the following precaution applies; A program must not transfer control between reading the first and second byte to another routine which also reads from that same Counter. Otherwise, an incorrect count will be read.

READ-BACK COMMAND

The third method uses the Read-Back command. This command allows the user to check the count value, programmed Mode, and current state of the OUT pin and Null Count flag of the selected counter(s).

The command is written into the Control Word Register and has the format shown in Figure 10. The command applies to the counters selected by setting their corresponding bits D3,D2,D1 = 1.

A0, A1 = 11 $\overline{CS}$ = 0 $\overline{RD}$ = 1 $\overline{WR}$ = 0

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	1	$\overline{COUNT}$	$\overline{STATUS}$	CNT 2	CNT 1	CNT 0	0

D_5: 0 = Latch count of selected counter(s)
D_4: 0 = Latch status of selected counter(s)
D_3: 1 = Select counter 2
D_2: 1 = Select counter 1
D_1: 1 = Select counter 0
D_0: Reserved for future expansion; must be 0

Figure 10. Read-Back Command Format

The read-back command may be used to latch multiple counter output latches (OL) by setting the $\overline{COUNT}$ bit D5 = 0 and selecting the desired counter(s). This single command is functionally equivalent to several counter latch commands, one for each counter latched. Each counter's latched count is held until it is read (or the counter is reprogrammed). That counter is automatically unlatched when read, but other counters remain latched until they are read. If multiple count read-back commands are issued to the same counter without reading the

count, all but the first are ignored; i.e., the count which will be read is the count at the time the first read-back command was issued.

The read-back command may also be used to latch status information of selected counter(s) by setting $\overline{\text{STATUS}}$ bit D4=0. Status must be latched to be read; status of a counter is accessed by a read from that counter.

The counter status format is shown in Figure 11. Bits D5 through D0 contain the counter's programmed Mode exactly as written in the last Mode Control Word. OUTPUT bit D7 contains the current state of the OUT pin. This allows the user to monitor the counter's output via software, possibly eliminating some hardware from a system.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
OUTPUT	NULL COUNT	RW1	RW0	M2	M1	M0	BCD

D_7 1 = Out Pin is 1
0 = Out Pin is 0
D_6 1 = Null count
0 = Count available for reading
D_5-D_0 Counter Programmed Mode (See Figure 7)

Figure 11. Status Byte

NULL COUNT bit D6 indicates when the last count written to the counter register (CR) has been loaded into the counting element (CE). The exact time this happens depends on the Mode of the counter and is described in the Mode Definitions, but until the count is loaded into the counting element (CE), it can't be read from the counter. If the count is latched or read before this time, the count value will not reflect the new count just written. The operation of Null Count is shown in Figure 12.

THIS ACTION:	CAUSES:
A. Write to the control word register:[1]	Null count=1
B. Write to the count register (CR);[2]	Null count=1
C. New count is loaded into CE (CR → CE);	Null count=0

[1] Only the counter specified by the control word will have its null count set to 1. Null count bits of other counters are unaffected.

[2] If the counter is programmed for two-byte counts (least significant byte then most significant byte) null count goes to 1 when the second byte is written.

Figure 12. Null Count Operation

If multiple status latch operations of the counter(s) are performed without reading the status, all but the first are ignored; i.e., the status that will be read is the status of the counter at the time the first status read-back command was issued.

Both count and status of the selected counter(s) may be latched simultaneously by setting both $\overline{\text{COUNT}}$ and $\overline{\text{STATUS}}$ bits D5,D4=0. This is functionally the same as issuing two separate read-back commands at once, and the above discussions apply here also. Specifically, if multiple count and/or status read-back commands are issued to the same counter(s) without any intervening reads, all but the first are ignored. This is illustrated in Figure 13.

If both count and status of a counter are latched, the first read operation of that counter will return latched status, regardless of which was latched first. The next one or two reads (depending on whether the counter is programmed for one or two type counts) return latched count. Subsequent reads return unlatched count.

Command D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Description	Results
1	1	0	0	0	0	1	0	Read back count and status of Counter 0	Count and status latched for Counter 0
1	1	1	0	0	1	0	0	Read back status of Counter 1	Status latched for Counter 1
1	1	1	0	1	1	0	0	Read back status of Counters 2, 1	Status latched for Counter 2, but not Counter 1
1	1	0	1	1	0	0	0	Read back count of Counter 2	Count latched for Counter 2
1	1	0	0	0	1	0	0	Read back count and status of Counter 1	Count latched for Counter 1, but not status
1	1	1	0	0	0	1	0	Read back status of Counter 1	Command ignored, status already latched for Counter 1

Figure 13. Read-Back Command Example

$\overline{CS}$	$\overline{RD}$	$\overline{WR}$	A_1	A_0	
0	1	0	0	0	Write into Counter 0
0	1	0	0	1	Write into Counter 1
0	1	0	1	0	Write into Counter 2
0	1	0	1	1	Write Control Word
0	0	1	0	0	Read from Counter 0
0	0	1	0	1	Read from Counter 1
0	0	1	1	0	Read from Counter 2
0	0	1	1	1	No-Operation (3-State)
1	X	X	X	X	No-Operation (3-State)
0	1	1	X	X	No-Operation (3-State)

Figure 14. Read/Write Operations Summary

Mode Definitions

The following are defined for use in describing the operation of the 82C54.

CLK PULSE: a rising edge, then a falling edge, in that order, of a Counter's CLK input.

TRIGGER: a rising edge of a Counter's GATE input.

COUNTER LOADING: the transfer of a count from the CR to the CE (refer to the "Functional Description")

MODE 0: INTERRUPT ON TERMINAL COUNT

Mode 0 is typically used for event counting. After the Control Word is written, OUT is initially low, and will remain low until the Counter reaches zero. OUT then goes high and remains high until a new count or a new Mode 0 Control Word is written into the Counter.

GATE = 1 enables counting; GATE = 0 disables counting. GATE has no effect on OUT.

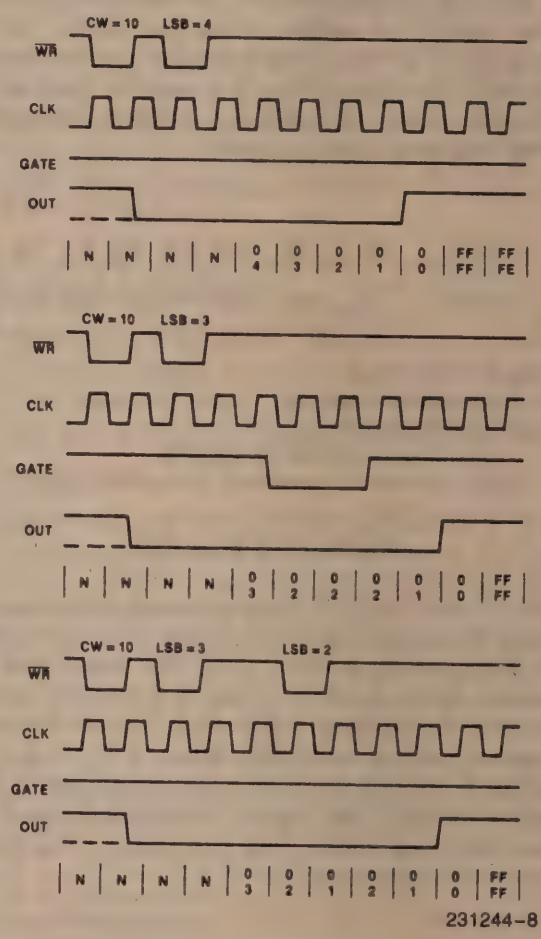
After the Control Word and initial count are written to a Counter, the initial count will be loaded on the next CLK pulse. This CLK pulse does not decrement the count, so for an initial count of N, OUT does not go high until N + 1 CLK pulses after the initial count is written.

If a new count is written to the Counter, it will be loaded on the next CLK pulse and counting will continue from the new count. If a two-byte count is written, the following happens:

1) Writing the first byte disables counting. OUT is set low immediately (no clock pulse required).
2) Writing the second byte allows the new count to be loaded on the next CLK pulse.

This allows the counting sequence to be synchronized by software. Again, OUT does not go high until N + 1 CLK pulses after the new count of N is written.

If an initial count is written while GATE = 0, it will still be loaded on the next CLK pulse. When GATE goes high, OUT will go high N CLK pulses later; no CLK pulse is needed to load the Counter as this has already been done.



NOTE:
The Following Conventions Apply To All Mode Timing Diagrams:
1. Counters are programmed for binary (not BCD) counting and for Reading/Writing least significant byte (LSB) only.
2. The counter is always selected ($\overline{CS}$ always low).
3. CW stands for "Control Word"; CW = 10 means a control word of 10, hex is written to the counter.
4. LSB stands for "Least Significant Byte" of count.
5. Numbers below diagrams are count values.
The lower number is the least significant byte.
The upper number is the most significant byte. Since the counter is programmed to Read/Write LSB only, the most significant byte cannot be read.
N stands for an undefined count.
Vertical lines show transitions between count values.

Figure 15. Mode 0

MODE 1: HARDWARE RETRIGGERABLE ONE-SHOT

OUT will be initially high. OUT will go low on the CLK pulse following a trigger to begin the one-shot pulse, and will remain low until the Counter reaches zero. OUT will then go high and remain high until the CLK pulse after the next trigger.

After writing the Control Word and initial count, the Counter is armed. A trigger results in loading the Counter and setting OUT low on the next CLK pulse, thus starting the one-shot pulse. An initial count of N will result in a one-shot pulse N CLK cycles in duration. The one-shot is retriggerable, hence OUT will remain low for N CLK pulses after any trigger. The one-shot pulse can be repeated without rewriting the same count into the counter. GATE has no effect on OUT.

If a new count is written to the Counter during a one-shot pulse, the current one-shot is not affected unless the Counter is retriggered. In that case, the Counter is loaded with the new count and the one-shot pulse continues until the new count expires.

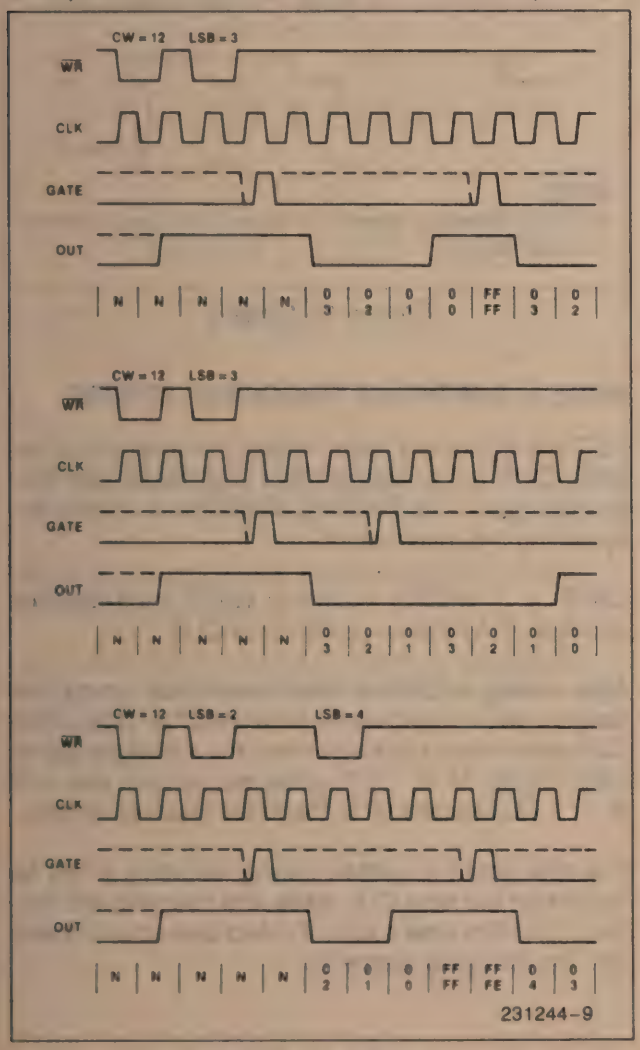


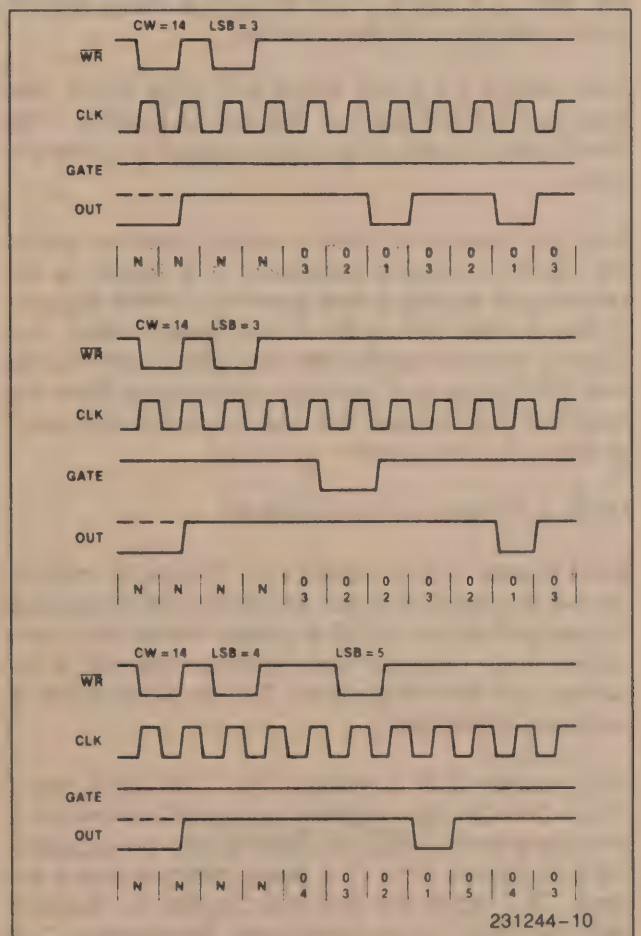
Figure 16. Mode 1

MODE 2: RATE GENERATOR

This Mode functions like a divide-by-N counter. It is typically used to generate a Real Time Clock interrupt. OUT will initially be high. When the initial count has decremented to 1, OUT goes low for one CLK pulse. OUT then goes high again, the Counter reloads the initial count and the process is repeated. Mode 2 is periodic; the same sequence is repeated indefinitely. For an initial count of N, the sequence repeats every N CLK cycles.

GATE = 1 enables counting; GATE = 0 disables counting. If GATE goes low during an output pulse, OUT is set high immediately. A trigger reloads the Counter with the initial count on the next CLK pulse; OUT goes low N CLK pulses after the trigger. Thus the GATE input can be used to synchronize the Counter.

After writing a Control Word and initial count, the Counter will be loaded on the next CLK pulse. OUT goes low N CLK Pulses after the initial count is written. This allows the Counter to be synchronized by software also.



NOTE:
A GATE transition should not occur one clock prior to terminal count.

Figure 17. Mode 2

Writing a new count while counting does not affect the current counting sequence. If a trigger is received after writing a new count but before the end of the current period, the Counter will be loaded with the new count on the next CLK pulse and counting will continue from the new count. Otherwise, the new count will be loaded at the end of the current counting cycle. In mode 2, a COUNT of 1 is illegal.

MODE 3: SQUARE WAVE MODE

Mode 3 is typically used for Baud rate generation. Mode 3 is similar to Mode 2 except for the duty cycle of OUT. OUT will initially be high. When half the initial count has expired, OUT goes low for the remainder of the count. Mode 3 is periodic; the sequence above is repeated indefinitely. An initial count of N results in a square wave with a period of N CLK cycles.

GATE = 1 enables counting; GATE = 0 disables counting. If GATE goes low while OUT is low, OUT is set high immediately; no CLK pulse is required. A trigger reloads the Counter with the initial count on the next CLK pulse. Thus the GATE input can be used to synchronize the Counter.

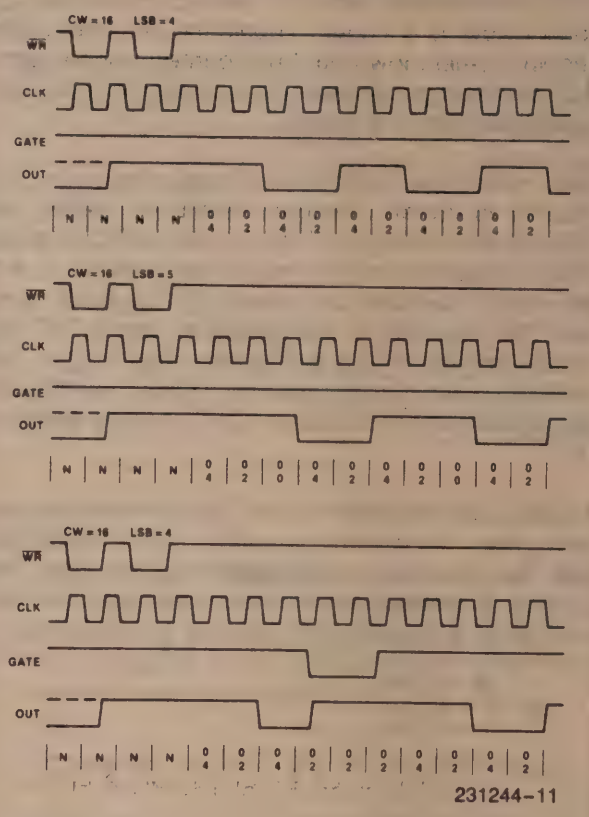
After writing a Control Word and initial count, the Counter will be loaded on the next CLK pulse. This allows the Counter to be synchronized by software also.

Writing a new count while counting does not affect the current counting sequence. If a trigger is received after writing a new count but before the end of the current half-cycle of the square wave, the Counter will be loaded with the new count on the next CLK pulse and counting will continue from the new count. Otherwise, the new count will be loaded at the end of the current half-cycle.

Mode 3 is implemented as follows:

Even counts: OUT is initially high. The initial count is loaded on one CLK pulse and then is decremented by two on succeeding CLK pulses. When the count expires OUT changes value and the Counter is reloaded with the initial count. The above process is repeated indefinitely.

Odd counts: OUT is initially high. The initial count minus one (an even number) is loaded on one CLK pulse and then is decremented by two on succeeding CLK pulses. One CLK pulse *after* the count expires, OUT goes low and the Counter is reloaded with the initial count minus one. Succeeding CLK pulses decrement the count by two. When the count expires, OUT goes high again and the Counter is reloaded with the initial count minus one. The above process is repeated indefinitely. So for odd counts, OUT will be high for (N + 1)/2 counts and low for (N − 1)/2 counts.



NOTE:
A GATE transition should not occur one clock prior to terminal count.

Figure 18. Mode 3

MODE 4: SOFTWARE TRIGGERED STROBE

OUT will be initially high. When the initial count expires, OUT will go low for one CLK pulse and then go high again. The counting sequence is "triggered" by writing the initial count.

GATE = 1 enables counting; GATE = 0 disables counting. GATE has no effect on OUT.

After writing a Control Word and initial count, the Counter will be loaded on the next CLK pulse. This CLK pulse does not decrement the count, so for an initial count of N, OUT does not strobe low until N + 1 CLK pulses after the initial count is written.

If a new count is written during counting, it will be loaded on the next CLK pulse and counting will continue from the new count. If a two-byte count is written, the following happens:

1) Writing the first byte has no effect on counting.
2) Writing the second byte allows the new count to be loaded on the next CLK pulse.

This allows the sequence to be "retriggered" by software. OUT strobes low N+1 CLK pulses after the new count of N is written.

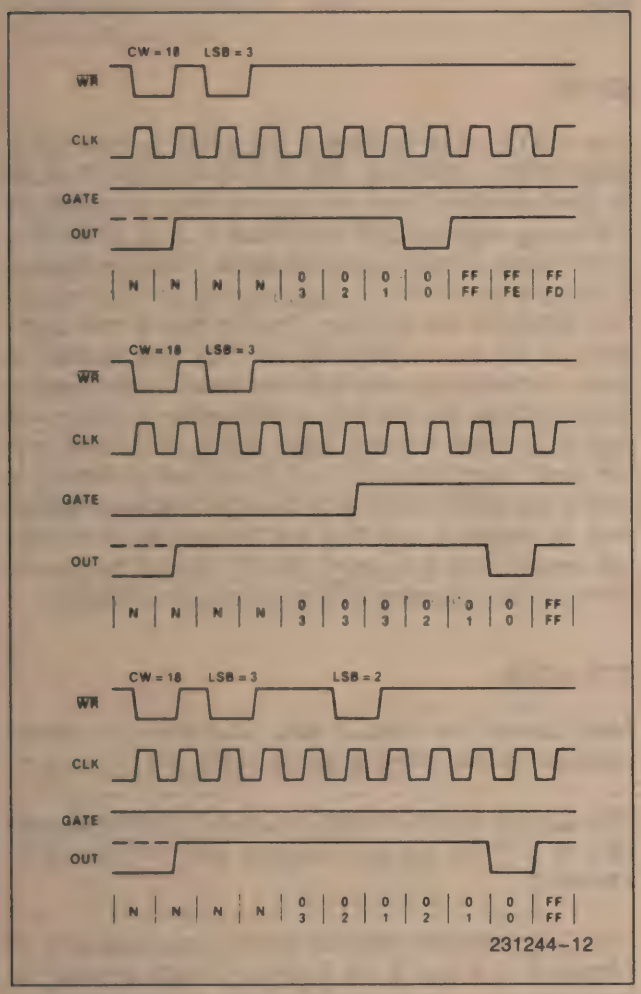


Figure 19. Mode 4

MODE 5: HARDWARE TRIGGERED STROBE (RETRIGGERABLE)

OUT will initially be high. Counting is triggered by a rising edge of GATE. When the initial count has expired, OUT will go low for one CLK pulse and then go high again.

After writing the Control Word and initial count, the counter will not be loaded until the CLK pulse after a trigger. This CLK pulse does not decrement the count, so for an initial count of N, OUT does not strobe low until N+1 CLK pulses after a trigger.

A trigger results in the Counter being loaded with the initial count on the next CLK pulse. The counting sequence is retriggerable. OUT will not strobe low for N + 1 CLK pulses after any trigger. GATE has no effect on OUT.

If a new count is written during counting, the current counting sequence will not be affected. If a trigger occurs after the new count is written but before the current count expires, the Counter will be loaded with the new count on the next CLK pulse and counting will continue from there.

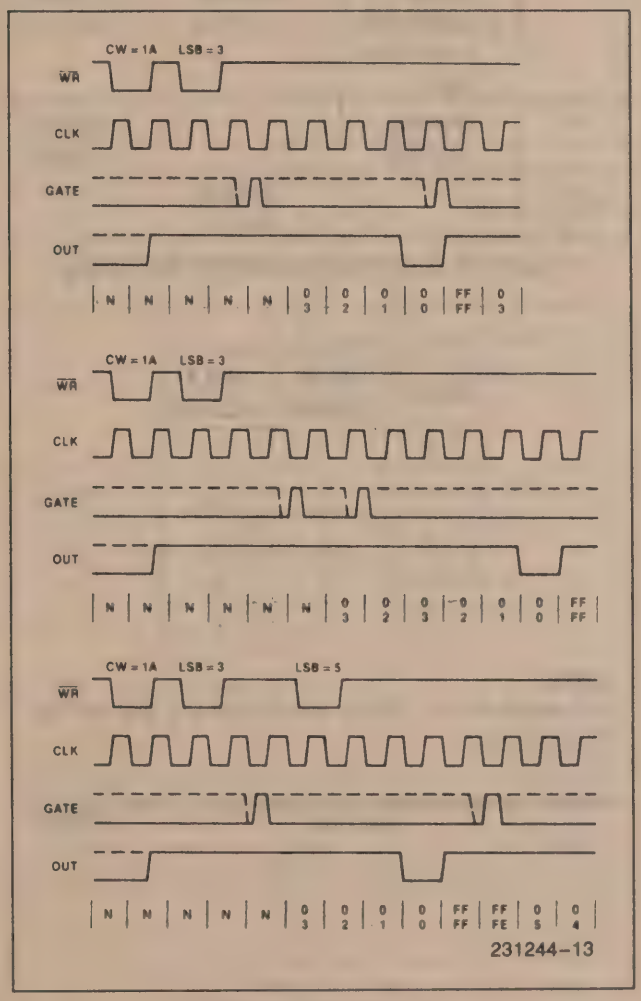


Figure 20. Mode 5

Signal Status Modes	Low Or Going Low	Rising	High
0	Disables counting	—	Enables counting
1	—	1) Initiates counting 2) Resets output after next clock	—
2	1) Disables counting 2) Sets output immediately high	Initiates counting	Enables counting
3	1) Disables counting 2) Sets output immediately high	Initiates counting	Enables counting
4	Disables counting	—	Enables counting
5	—	Initiates counting	—

Figure 21. Gate Pin Operations Summary

MODE	MIN COUNT	MAX COUNT
0	1	0
1	1	0
2	2	0
3	2	0
4	1	0

NOTE:
0 is equivalent to 2^{16} for binary counting and 10^4 for BCD counting

Figure 22. Minimum and Maximun initial Counts

Operation Common to All Modes

Programming

When a Control Word is written to a Counter, all Control Logic is immediately reset and OUT goes to a known initial state; no CLK pulses are required for this.

GATE

The GATE input is always sampled on the rising edge of CLK. In Modes 0, 2, 3, and 4 the GATE input is level sensitive, and the logic level is sampled on the rising edge of CLK. In Modes 1, 2, 3, and 5 the GATE input is rising-edge sensitive. In these Modes, a rising edge of GATE (trigger) sets an edge-sensitive flip-flop in the Counter. This flip-flop is then sampled on the next rising edge of CLK; the flip-flop is reset immediately after it is sampled. In this way, a trigger will be detected no matter when it occurs—a high logic level does not have to be maintained until the next rising edge of CLK. Note that in Modes 2 and 3, the GATE input is both edge- and level-sensitive. In Modes 2 and 3, if a CLK source other than the system clock is used, GATE should be pulsed immediately following $\overline{WR}$ of a new count value.

COUNTER

New counts are loaded and Counters are decremented on the falling edge of CLK.

The largest possible initial count is 0; this is equivalent to 2^{16} for binary counting and 10^4 for BCD counting.

The Counter does not stop when it reaches zero. In Modes 0, 1, 4, and 5 the Counter "wraps around" to the highest count, either FFFF hex for binary counting or 9999 for BCD counting, and continues counting. Modes 2 and 3 are periodic; the Counter reloads itself with the initial count and continues counting from there.

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias.....0°C to 70°C
Storage Temperature .....−65° to +150°C
Supply Voltage .....−0.5 to +8.0V
Operating Voltage .....+4V to +7V
Voltage on any Input.....GND −2V to +6.5V
Voltage on any Output ..GND−0.5V to V_{CC} + 0.5V
Power Dissipation .....1 Watt

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

NOTICE: Specifications contained within the following tables are subject to change.

D.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = 5V ± 10%, GND = 0V)

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{IL}	Input Low Voltage	−0.5	0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC} + 0.5	V	
V_{OL}	Output Low Voltage		0.4	V	I_{OL} = 2.5 mA
V_{OH}	Output High Voltage	3.0 V_{CC} − 0.4		V V	I_{OH} = −2.5 mA I_{OH} = −100 μA
I_{IL}	Input Load Current		±10	V	V_{IN} = V_{CC} to 0V
I_{OFL}	Output Float Leakage Current		±10	μA	V_{OUT} = V_{CC} to 0.45V
I_{CC}	V_{CC} Supply Current		10	mA	Clk Freq = 8MHz 82C54 10MHz 82C54-2
I_{CCSB}	V_{CC} Supply Current-Standby		10	μA	CLK Freq = DC $\overline{CS}$ = HIGH All Inputs/Data Bus HIGH All Outputs Floating

CAPACITANCE (T_A = 25°C, V_{CC} = GND = 0V)

Symbol	Parameter	Min	Max	Units	Test Conditions
C_{IN}	Input Capacitance		10	pF	f_c = 1 MHz
$C_{I/O}$	I/O Capacitance		20	pF	Unmeasured pins returned to GND
C_{OUT}	Output Capacitance		20	pF	

A.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = 5V ±10%, GND = 0V)

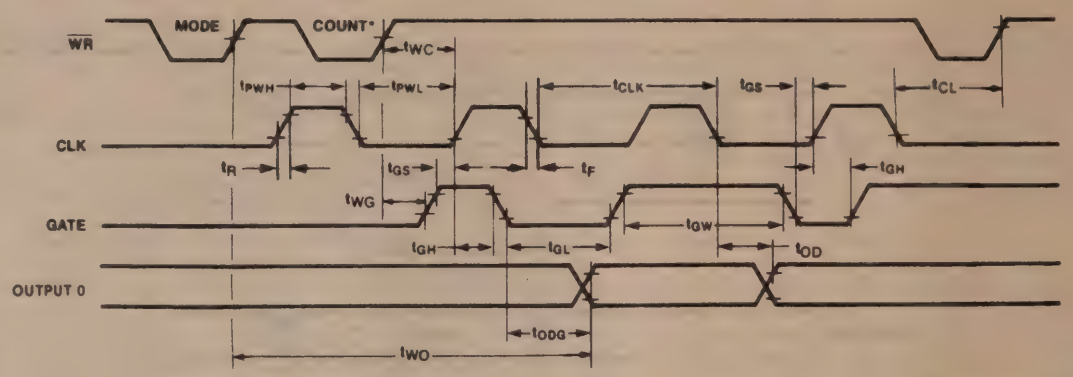
BUS PARAMETERS (Note 1)

READ CYCLE

Symbol	Parameter	82C54		82C54-2		Units
		Min	Max	Min	Max	
t_{AR}	Address Stable Before $\overline{RD}$ ↓	45		30		ns
t_{SR}	$\overline{CS}$ Stable Before $\overline{RD}$ ↓	0		0		ns
t_{RA}	Address Hold Time After $\overline{RD}$ ↓	0		0		ns
t_{RR}	$\overline{RD}$ Pulse Width	150		95		ns
t_{RD}	Data Delay from $\overline{RD}$ ↓		120		85	ns
t_{AD}	Data Delay from Address		220		185	ns
t_{DF}	$\overline{RD}$ ↑ to Data Floating	5	90	5	65	ns
t_{RV}	Command Recovery Time	200		165		ns

NOTE:
1. AC timings measured at V_{OH} = 2.0V, V_{OL} = 0.8V.

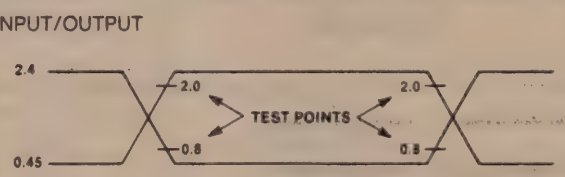
CLOCK AND GATE



231244-17

* Last byte of count being written

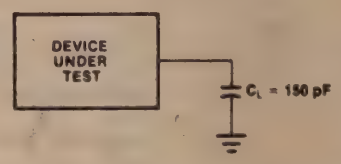
A.C. TESTING INPUT, OUTPUT WAVEFORM



231244-18

A.C. Testing: Inputs are driven at 2.4V for a logic "1" and 0.45V for a logic "0." Timing measurements are made at 2.0V for a logic "1" and 0.8V for a logic "0."

A.C. TESTING LOAD CIRCUIT



231244-19

C_L = 150 pF

C_L includes jig capacitance

8255A/8255A-5
PROGRAMMABLE PERIPHERAL INTERFACE

- **MCS-85™ Compatible 8255A-5**
- **24 Programmable I/O Pins**
- **Completely TTL Compatible**
- **Fully Compatible with Intel® Microprocessor Families**
- **Improved Timing Characteristics**
- **Direct Bit Set/Reset Capability Easing Control Application Interface**
- **Reduces System Package Count**
- **Improved DC Driving Capability**
- **Available in EXPRESS**
  —Standard Temperature Range
  —Extended Temperature Range

The Intel® 8255A is a general purpose programmable I/O device designed for use with Intel® microprocessors. It has 24 I/O pins which may be individually programmed in 2 groups of 12 and used in 3 major modes of operation. In the first mode (MODE 0), each group of 12 I/O pins may be programmed in sets of 4 to be input or output. In MODE 1, the second mode, each group may be programmed to have 8 lines of input or output. Of the remaining 4 pins, 3 are used for handshaking and interrupt control signals. The third mode of operation (MODE 2) is a bidirectional bus mode which uses 8 lines for a bidirectional bus, and 5 lines, borrowing one from the other group, for handshaking.

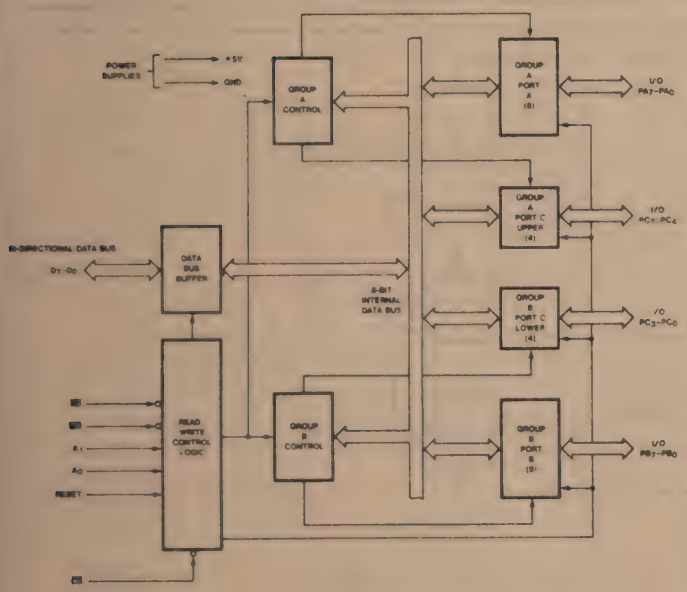


Figure 1. 8255A Block Diagram

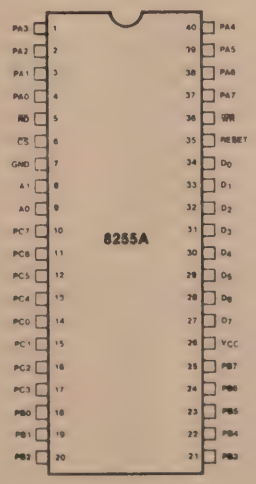


Figure 2. Pin Configuration

©INTEL CORPORATION 1982

Order Number: 231308-001

8255A FUNCTIONAL DESCRIPTION

General

The 8255A is a programmable peripheral interface (PPI) device designed for use in Intel® microcomputer systems. Its function is that of a general purpose I/O component to interface peripheral equipment to the microcomputer system bus. The functional configuration of the 8255A is programmed by the system software so that normally no external logic is necessary to interface peripheral devices or structures.

Data Bus Buffer

This 3-state bidirectional 8-bit buffer is used to interface the 8255A to the system data bus. Data is transmitted or received by the buffer upon execution of input or output instructions by the CPU. Control words and status information are also transferred through the data bus buffer.

Read/Write and Control Logic

The function of this block is to manage all of the internal and external transfers of both Data and Control or Status words. It accepts inputs from the CPU Address and Control busses and in turn, issues commands to both of the Control Groups.

($\overline{CS}$)

Chip Select. A "low" on this input pin enables the communiction between the 8255A and the CPU.

($\overline{RD}$)

Read. A "low" on this input pin enables the 8255A to send the data or status information to the CPU on the data bus. In essence, it allows the CPU to "read from" the 8255A.

($\overline{WR}$)

Write. A "low" on this input pin enables the CPU to write data or control words into the 8255A.

(A_0 and A_1)

Port Select 0 and Port Select 1. These input signals, in conjunction with the RD and WR inputs, control the selection of one of the three ports or the control word registers. They are normally connected to the least significant bits of the address bus (A_0 and A_1).

8255A BASIC OPERATION

A_1	A_0	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	INPUT OPERATION (READ)
0	0	0	1	0	PORT A → DATA BUS
0	1	0	1	0	PORT B → DATA BUS
1	0	0	1	0	PORT C → DATA BUS
					OUTPUT OPERATION (WRITE)
0	0	1	0	0	DATA BUS → PORT A
0	1	1	0	0	DATA BUS → PORT B
1	0	1	0	0	DATA BUS → PORT C
1	1	1	0	0	DATA BUS → CONTROL
					DISABLE FUNCTION
X	X	X	X	1	DATA BUS → 3-STATE
1	1	0	1	0	ILLEGAL CONDITION
X	X	1	1	0	DATA BUS → 3-STATE

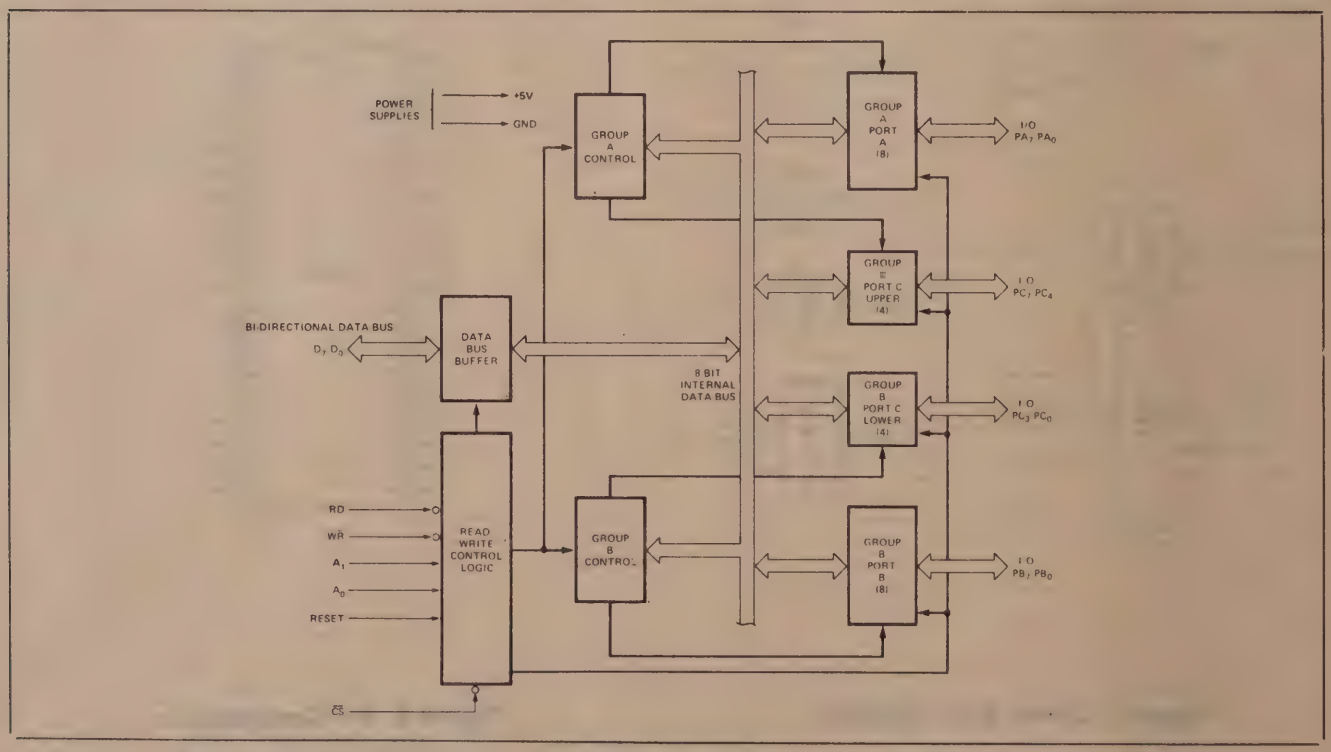


Figure 3. 8255A Block Diagram Showing Data Bus Buffer and Read/Write Control Logic Functions

231308-001

(RESET)

Reset. A "high" on this input clears the control register and all ports (A, B, C) are set to the input mode.

Group A and Group B Controls

The functional configuration of each port is programmed by the systems software. In essence, the CPU "outputs" a control word to the 8255A. The control word contains information such as "mode", "bit set", "bit reset", etc., that initializes the functional configuration of the 8255A.

Each of the Control blocks (Group A and Group B) accepts "commands" from the Read/Write Control Logic, receives "control words" from the internal data bus and issues the proper commands to its associated ports.

Control Group A – Port A and Port C upper (C7-C4)
Control Group B – Port B and Port C lower (C3-C0)

The Control Word Register can **Only** be written into. No Read operation of the Control Word Register is allowed.

Ports A, B, and C

The 8255A contains three 8-bit ports (A, B, and C). All can be configured in a wide variety of functional characteristics by the system software but each has its own special features or "personality" to further enhance the power and flexibility of the 8255A.

Port A. One 8-bit data output latch/buffer and one 8-bit data input latch.

Port B. One 8-bit data input/output latch/buffer and one 8-bit data input buffer.

Port C. One 8-bit data output latch/buffer and one 8-bit data input buffer (no latch for input). This port can be divided into two 4-bit ports under the mode control. Each 4-bit port contains a 4-bit latch and it can be used for the control signal outputs and status signal inputs in conjunction with ports A and B.

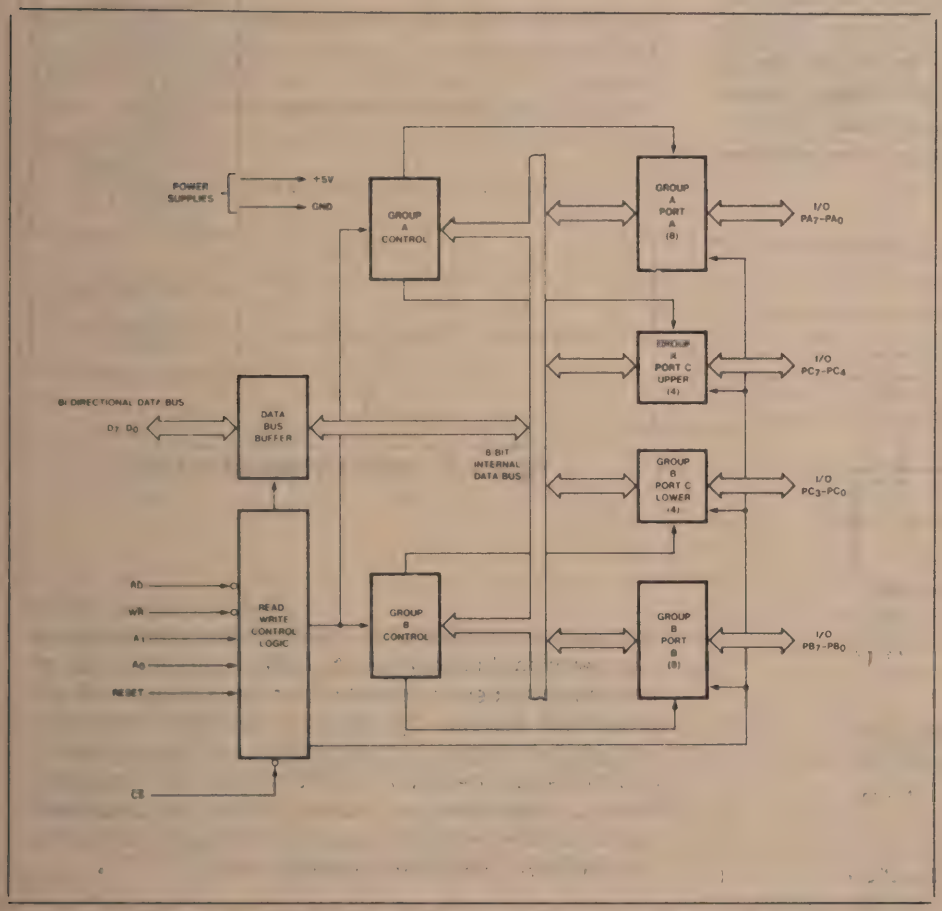
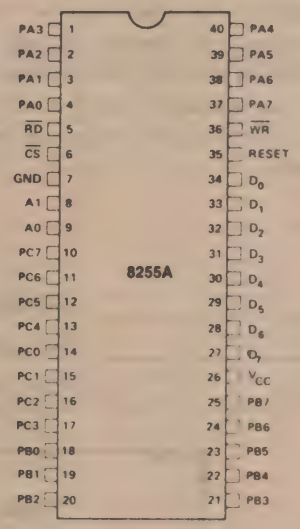


Figure 4. 8225A Block Diagram Showing Group A and Group B Control Functions

PIN CONFIGURATION



PIN NAMES

D_7 D_0	DATA BUS (BI DIRECTIONAL)
RESET	RESET INPUT
$\overline{CS}$	CHIP SELECT
$\overline{RD}$	READ INPUT
$\overline{WR}$	WRITE INPUT
A0, A1	PORT ADDRESS
PA7-PA0	PORT A (BIT)
PB7-PB0	PORT B (BIT)
PC7-PC0	PORT C (BIT)
V_{CC}	+5 VOLTS
GND	0 VOLTS

231308-001

8255A OPERATIONAL DESCRIPTION

Mode Selection

There are three basic modes of operation that can be selected by the system software:

Mode 0 — Basic Input/Output
Mode 1 — Strobed Input/Output
Mode 2 — Bi-Directional Bus

When the reset input goes "high" all ports will be set to the input mode (i.e., all 24 lines will be in the high impedance state). After the reset is removed the 8255A can remain in the input mode with no additional initialization required. During the execution of the system program any of the other modes may be selected using a single output instruction. This allows a single 8255A to service a variety of peripheral devices with a simple software maintenance routine.

The modes for Port A and Port B can be separately defined, while Port C is divided into two portions as required by the Port A and Port B definitions. All of the output registers, including the status flip-flops, will be reset whenever the mode is changed. Modes may be combined so that their functional definition can be "tailored" to almost any I/O structure. For instance; Group B can be programmed in Mode 0 to monitor simple switch closings or display computational results, Group A could be programmed in Mode 1 to monitor a keyboard or tape reader on an interrupt-driven basis.

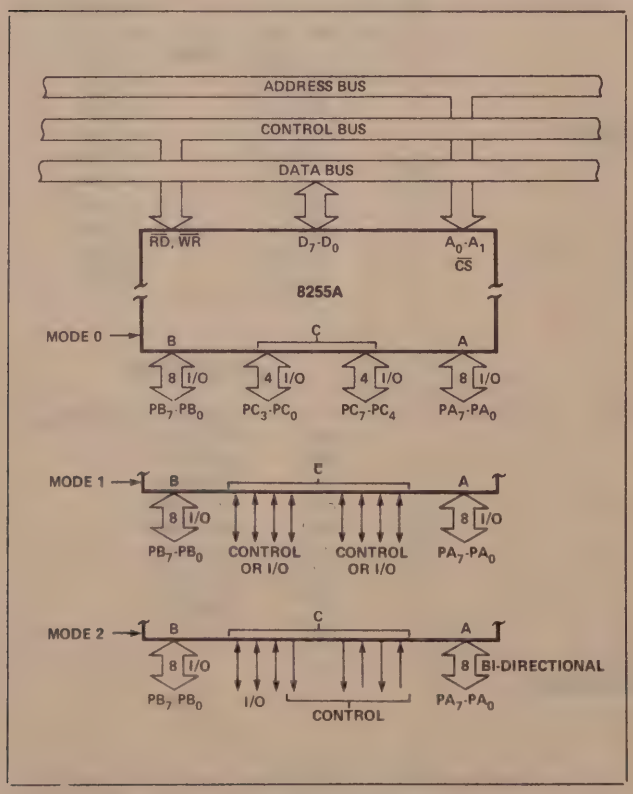


Figure 5. Basic Mode Definitions and Bus Interface

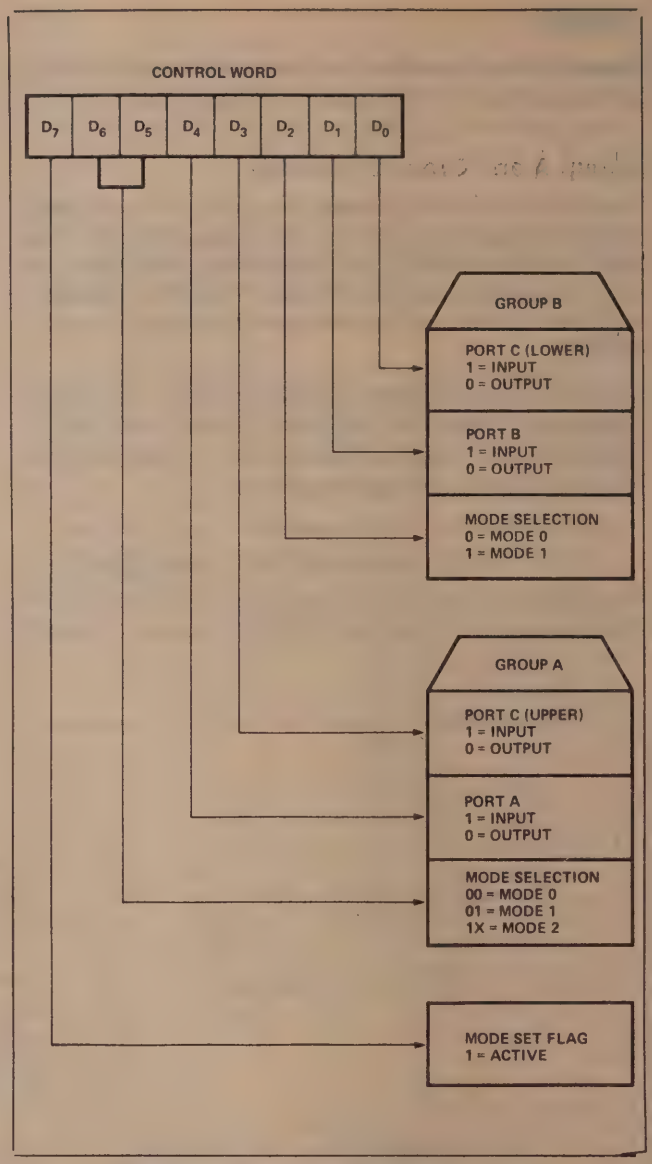


Figure 6. Mode Definition Format

The mode definitions and possible mode combinations may seem confusing at first but after a cursory review of the complete device operation a simple, logical I/O approach will surface. The design of the 8255A has taken into account things such as efficient PC board layout, control signal definition vs PC layout and complete functional flexibility to support almost any peripheral device with no external logic. Such design represents the maximum use of the available pins.

Single Bit Set/Reset Feature

Any of the eight bits of Port C can be Set or Reset using a single OUTput instruction. This feature reduces software requirements in Control-based applications.

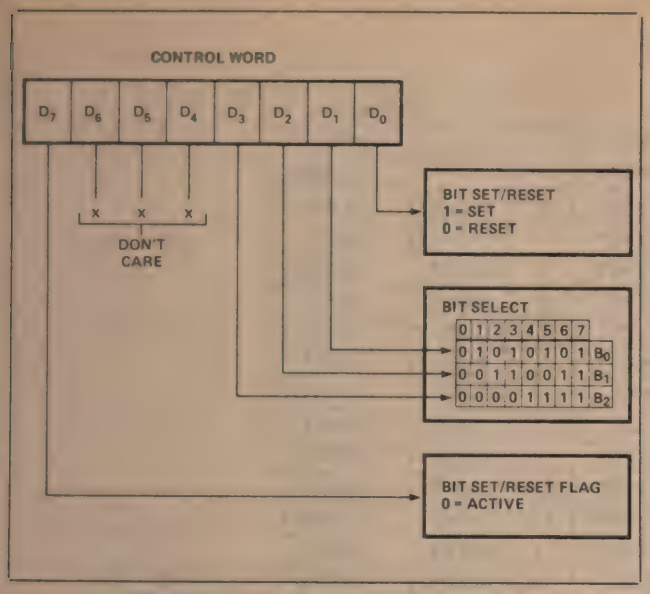


Figure 7. Bit Set/Reset Format

When Port C is being used as status/control for Port A or B, these bits can be set or reset by using the Bit Set/Reset operation just as if they were data output ports.

Interrupt Control Functions

When the 8255A is programmed to operate in mode 1 or mode 2, control signals are provided that can be used as interrupt request inputs to the CPU. The interrupt request signals, generated from port C, can be inhibited or enabled by setting or resetting the associated INTE flip-flop, using the bit set/reset function of port C.

This function allows the Programmer to disallow or allow a specific I/O device to interrupt the CPU without affecting any other device in the interrupt structure.

INTE flip-flop definition:

(BIT-SET) — INTE is SET — Interrupt enable
(BIT-RESET) — INTE is RESET — Interrupt disable

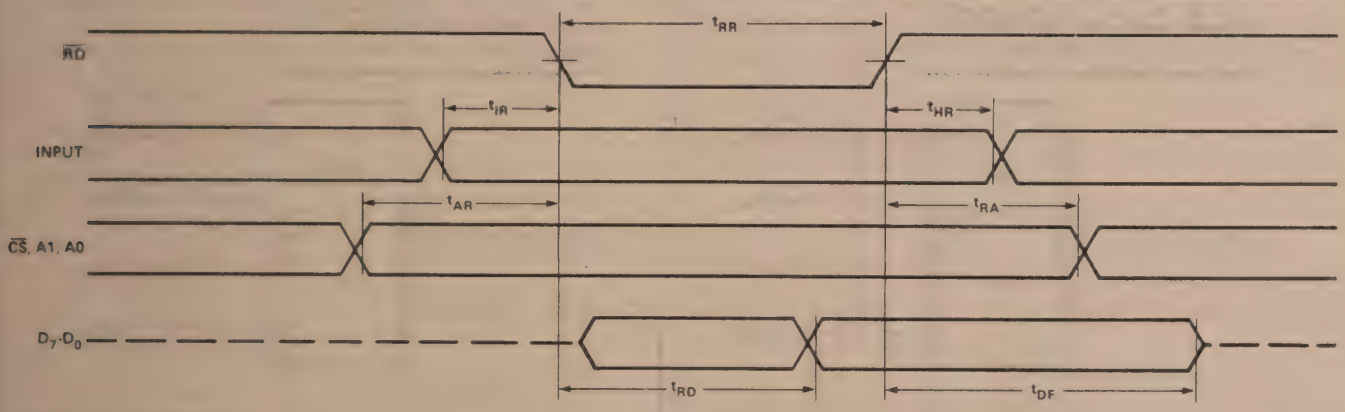
Note: All Mask flip-flops are automatically reset during mode selection and device Reset.

Operating Modes

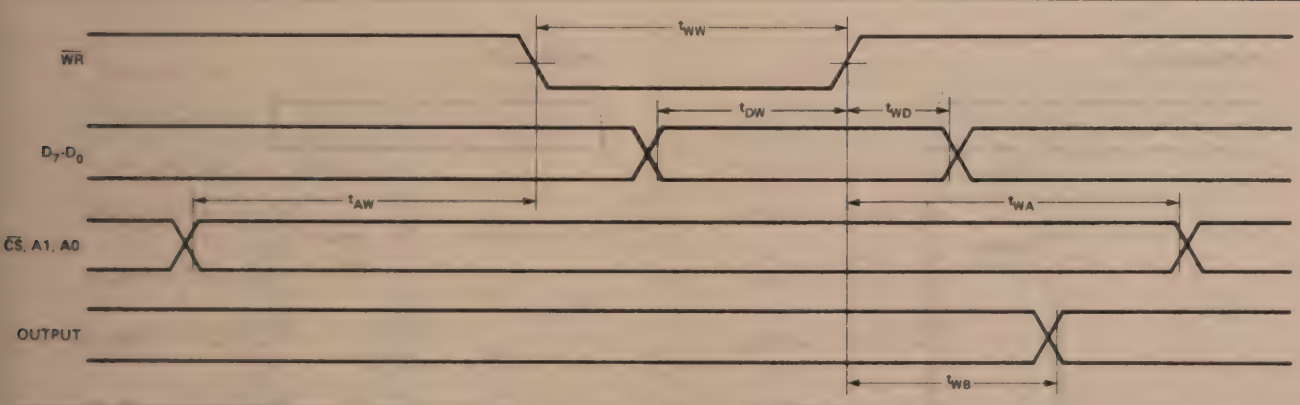
MODE 0 (Basic Input/Output). This functional configuration provides simple input and output operations for each of the three ports. No "handshaking" is required, data is simply written to or read from a specified port.

Mode 0 Basic Functional Definitions:

- Two 8-bit ports and two 4-bit ports.
- Any port can be input or output.
- Outputs are latched.
- Inputs are not latched.
- 16 different Input/Output configurations are possible in this Mode.



MODE 0 (Basic Input)



MODE 0 (Basic Output)

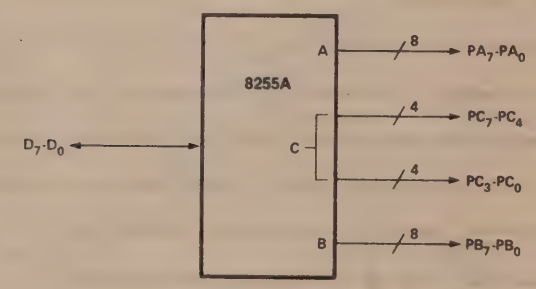
MODE 0 Port Definition

A		B		GROUP A			GROUP B	
D_4	D_3	D_1	D_0	PORT A	PORT C (UPPER)	#	PORT B	PORT C (LOWER)
0	0	0	0	OUTPUT	OUTPUT	0	OUTPUT	OUTPUT
0	0	0	1	OUTPUT	OUTPUT	1	OUTPUT	INPUT
0	0	1	0	OUTPUT	OUTPUT	2	INPUT	OUTPUT
0	0	1	1	OUTPUT	OUTPUT	3	INPUT	INPUT
0	1	0	0	OUTPUT	INPUT	4	OUTPUT	OUTPUT
0	1	0	1	OUTPUT	INPUT	5	OUTPUT	INPUT
0	1	1	0	OUTPUT	INPUT	6	INPUT	OUTPUT
0	1	1	1	OUTPUT	INPUT	7	INPUT	INPUT
1	0	0	0	INPUT	OUTPUT	8	OUTPUT	OUTPUT
1	0	0	1	INPUT	OUTPUT	9	OUTPUT	INPUT
1	0	1	0	INPUT	OUTPUT	10	INPUT	OUTPUT
1	0	1	1	INPUT	OUTPUT	11	INPUT	INPUT
1	1	0	0	INPUT	INPUT	12	OUTPUT	OUTPUT
1	1	0	1	INPUT	INPUT	13	OUTPUT	INPUT
1	1	1	0	INPUT	INPUT	14	INPUT	OUTPUT
1	1	1	1	INPUT	INPUT	15	INPUT	INPUT

MODE 0 Configurations

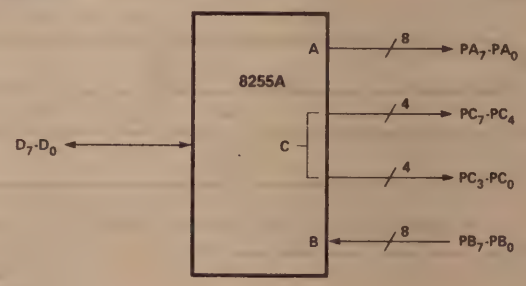
CONTROL WORD #0

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	0	0	0	0



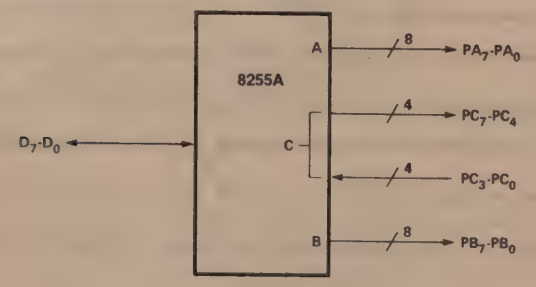
CONTROL WORD #2

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	0	0	1	0



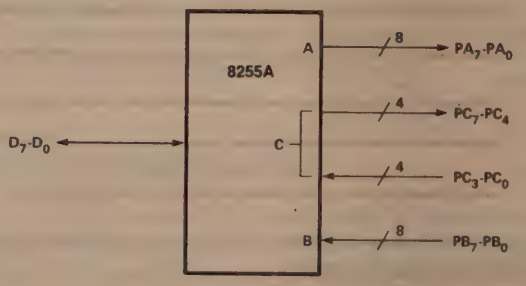
CONTROL WORD #1

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	0	0	0	1



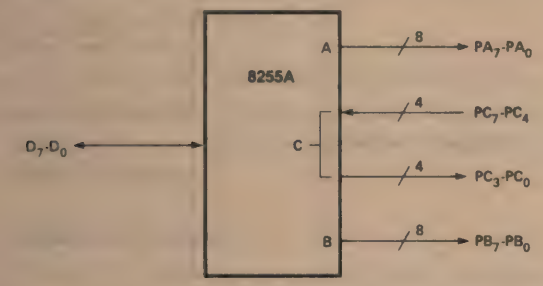
CONTROL WORD #3

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	0	0	1	1



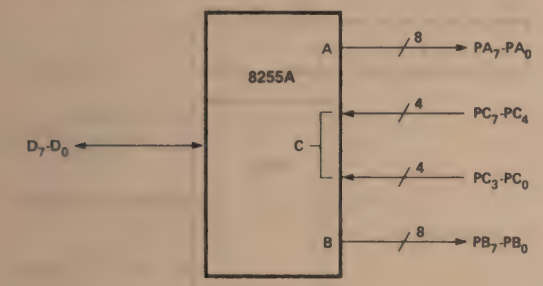
CONTROL WORD #4

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	1	0	0	0



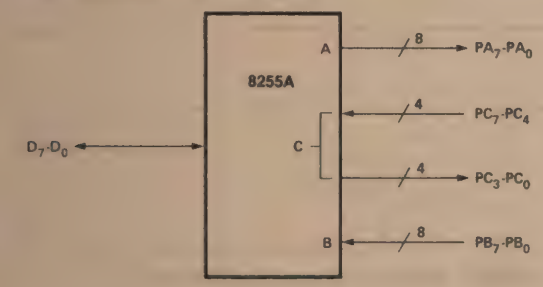
CONTROL WORD #5

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	1	0	0	1



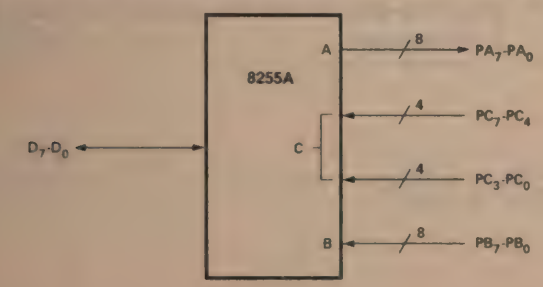
CONTROL WORD #6

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	1	0	1	0



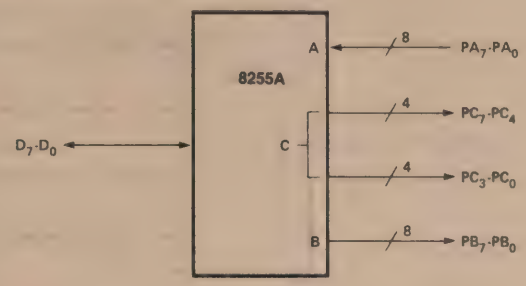
CONTROL WORD #7

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	1	0	1	1



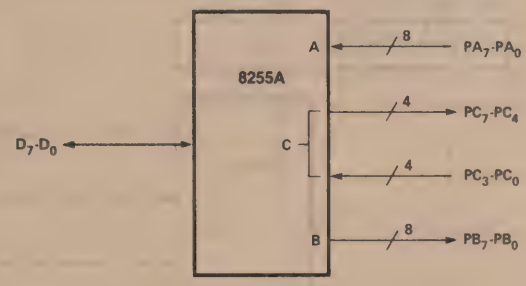
CONTROL WORD #8

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	0	0	0	0



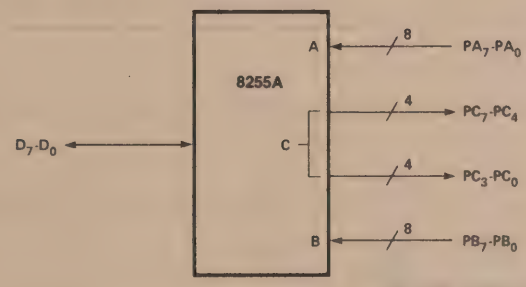
CONTROL WORD #9

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	0	0	0	1



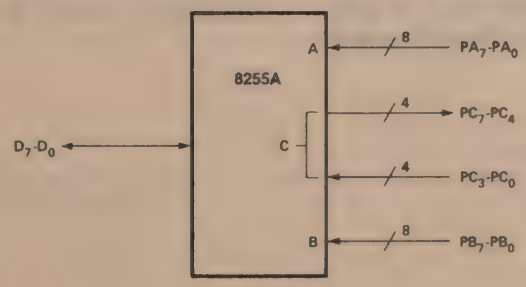
CONTROL WORD #10

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	0	0	1	0



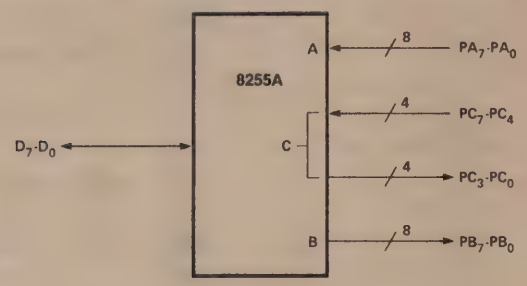
CONTROL WORD #11

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	0	0	1	1



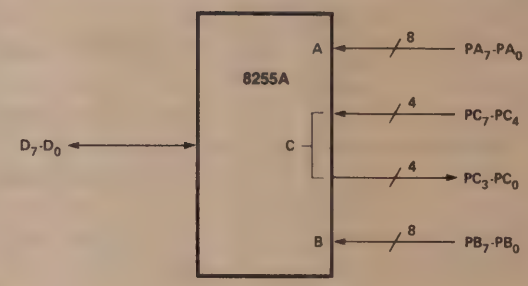
CONTROL WORD #12

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	1	0	0	0



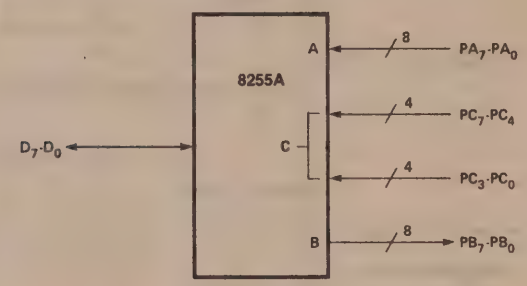
CONTROL WORD #14

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	1	0	1	0



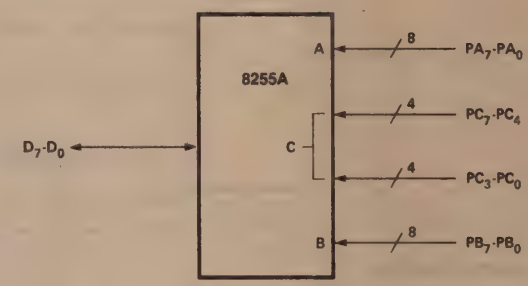
CONTROL WORD #13

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	1	0	0	1



CONTROL WORD #15

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	1	0	1	1



Operating Modes

MODE 1 (Strobed Input/Output). This functional configuration provides a means for transferring I/O data to or from a specified port in conjunction with strobes or "handshaking" signals. In mode 1, port A and port B use the lines on port C to generate or accept these "hand-shaking" signals.

Mode 1 Basic Functional Definitions:

- Two Groups (Group A and Group B)
- Each group contains one 8-bit data port and one 4-bit control/data port.
- The 8-bit data port can be either input or output. Both inputs and outputs are latched.
- The 4-bit port is used for control and status of the 8-bit data port.

Input Control Signal Definition

$\overline{STB}$ (Strobe Input). A "low" on this input loads data into the input latch.

IBF (Input Buffer Full F/F)

A "high" on this output indicates that the data has been loaded into the input latch; in essence, an acknowledgement. IBF is set by STB input being low and is reset by the rising edge of the RD input.

INTR (Interrupt Request)

A "high" on this output can be used to interrupt the CPU when an input device is requesting service. INTR is set by the $\overline{STB}$ is a "one", IBF is a "one" and INTE is a "one". It is reset by the falling edge of $\overline{RD}$. This procedure allows an input device to request service from the CPU by simply strobing its data into the port.

INTE A

Controlled by bit set/reset of PC_4.

INTE B

Controlled by bit set/reset of PC_2.

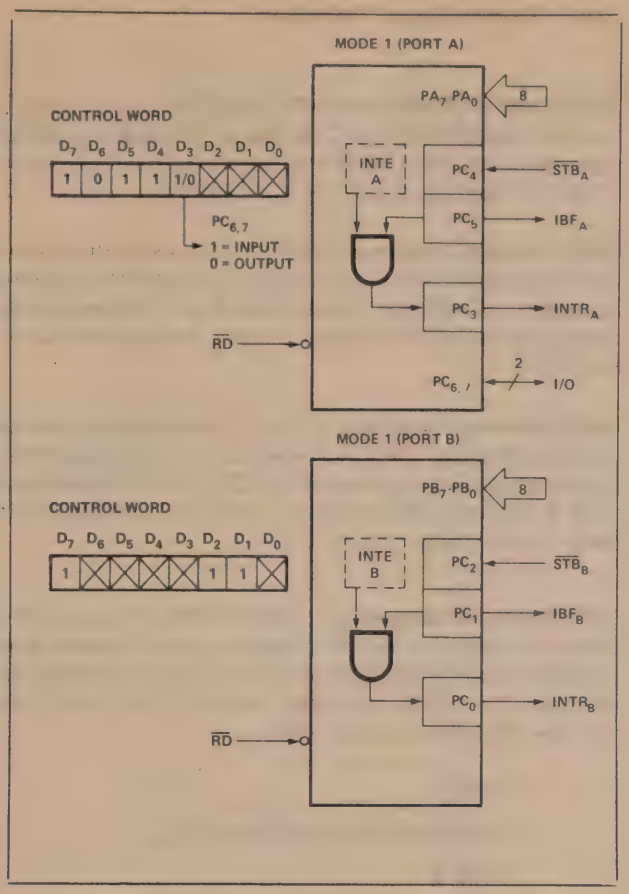


Figure 8. MODE 1 Input

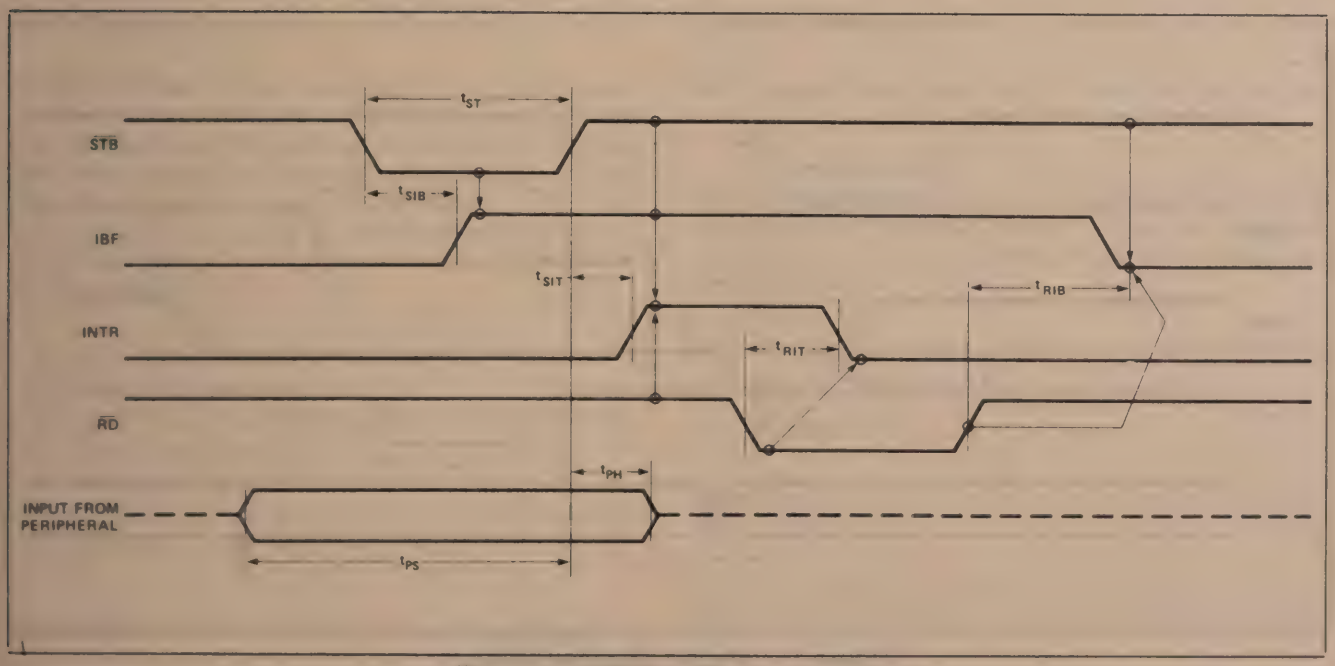


Figure 9. MODE 1 (Strobed Input)

Output Control Signal Definition

$\overline{OBF}$ (Output Buffer Full F/F). The $\overline{OBF}$ output will go "low" to indicate that the CPU has written data out to the specified port. The $\overline{OBF}$ F/F will be set by the rising edge of the $\overline{WR}$ input and reset by $\overline{ACK}$ input being low.

$\overline{ACK}$ (Acknowledge Input). A "low" on this input informs the 8255A that the data from port A or port B has been accepted. In essence, a response from the peripheral device indicating that it has recieved the data output by the CPU.

INTR (Interrupt Request). A "high" on this output can be used to interrupt the CPU when an output device has accepted data transmitted by the CPU. INTR is set when $\overline{ACK}$ is a "one", $\overline{OBF}$ is a "one", and INTE is a "one". It is reset by the falling edge of $\overline{WR}$.

INTR (Interrupt Request). A "high" on this output can be used to interrupt the CPU when an output device has accepted data transmitted by the CPU. INTR is set when $\overline{ACK}$ is a "one", $\overline{OBF}$ is a "one", and INTE is a "one". It is reset by the falling edge of WR.

INTE A

Controlled by bit set/reset of PC_6.

INTE B

Controlled by bit set/reset of PC_2.

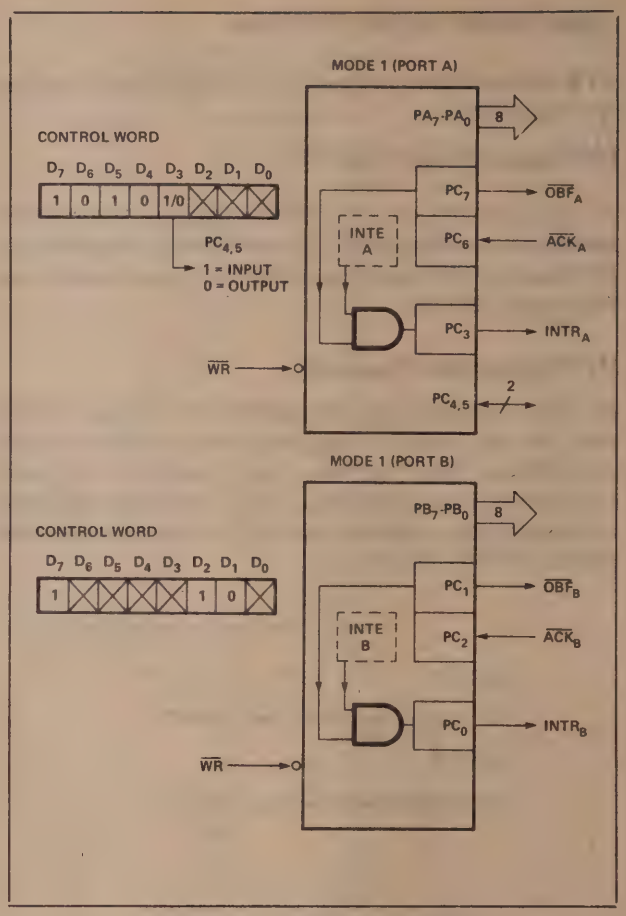


Figure 10. MODE 1 Output

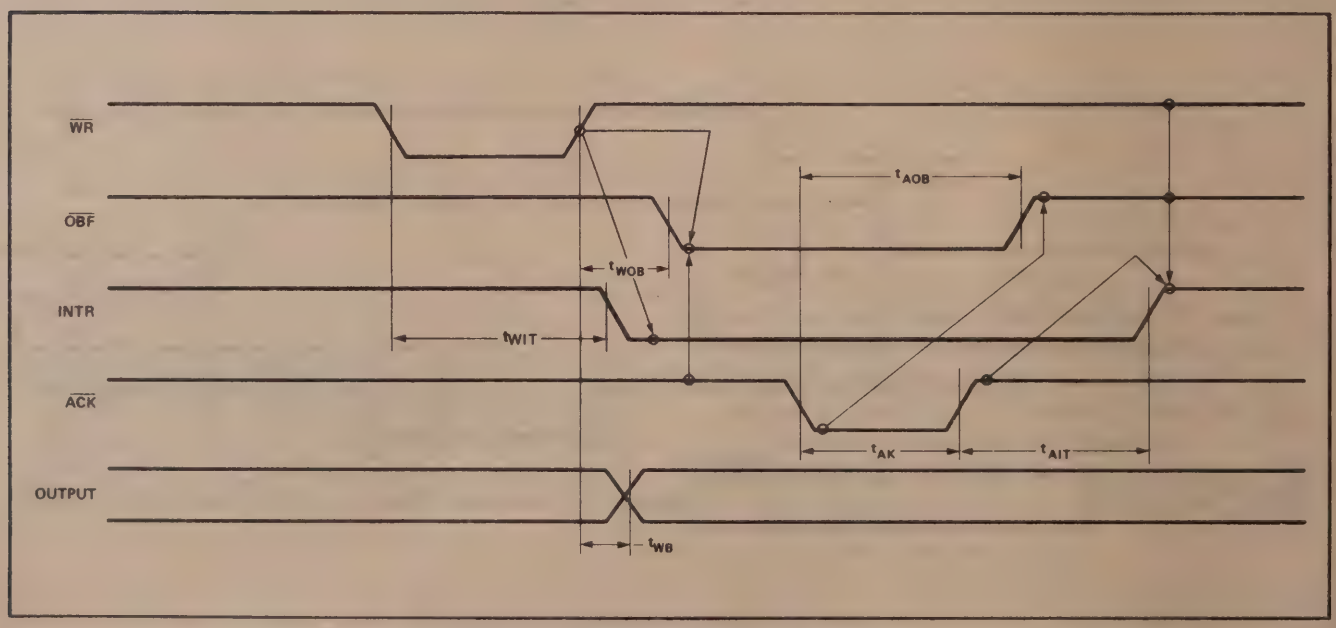


Figure 11. Mode 1 (Strobed Output)

Combinations of MODE 1

Port A and Port B can be individually defined as input or output in Mode 1 to support a wide variety of strobed I/O applications.

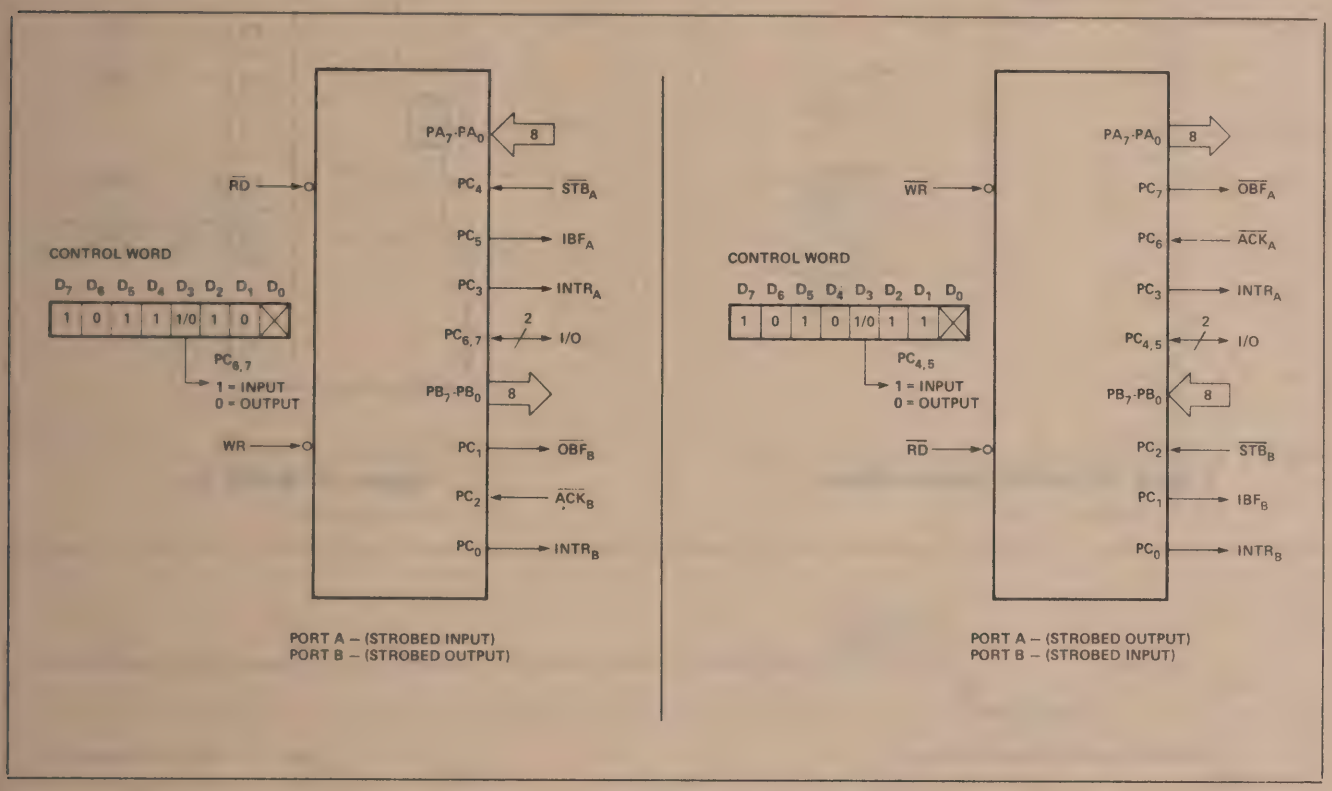


Figure 12. Combinations of MODE 1

Operating Modes

MODE 2 (Strobed Bidirectional Bus I/O). This functional configuration provides a means for communicating with a peripheral device or structure on a single 8-bit bus for both transmitting and receiving data (bidirectional bus I/O). "Handshaking" signals are provided to maintain proper bus flow discipline in a similar manner to MODE 1. Interrupt generation and enable/disable functions are also available.

MODE 2 Basic Functional Definitions:

- Used in Group A only.
- One 8-bit, bi-directional bus Port (Port A) and a 5-bit control Port (Port C).
- Both inputs and outputs are latched.
- The 5-bit control port (Port C) is used for control and status for the 8-bit, bi-directional bus port (Port A).

Bidirectional Bus I/O Control Signal Definition

INTR (Interrupt Request). A high on this output can be used to interrupt the CPU for both input or output operations.

Output Operations

$\overline{OBF}$ (Output Buffer Full). The $\overline{OBF}$ output will go "low" to indicate that the CPU has written data out to port A.

$\overline{ACK}$ (Acknowledge). A "low" on this input enables the tri-state output buffer of port A to send out the data. Otherwise, the output buffer will be in the high impedance state.

INTE 1 (The INTE Flip-Flop Associated with OBF). Controlled by bit set/reset of PC_6.

Input Operations

$\overline{STB}$ (Strobe Input). A "low" on this input loads data into the input latch.

IBF (Input Buffer Full F/F). A "high" on this output indicates that data has been loaded into the input latch.

INTE 2 (The INTE Flip-Flop Associated with IBF). Controlled by bit set/reset of PC_4.

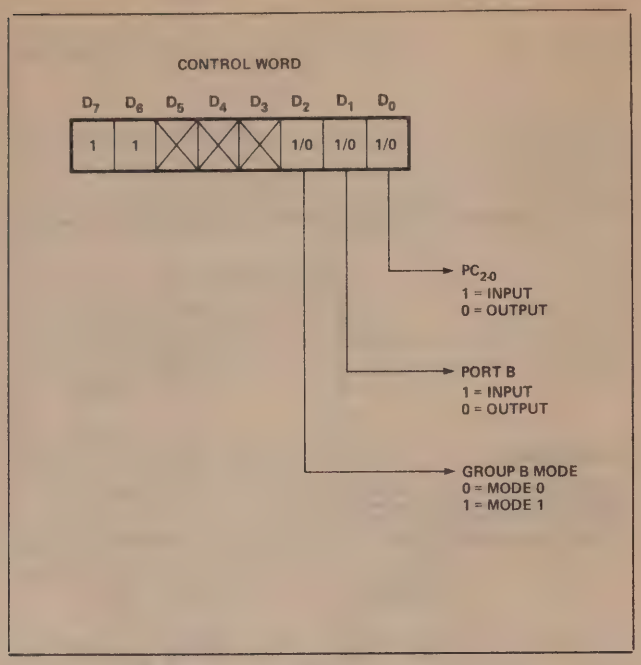


Figure 13. MODE Control Word

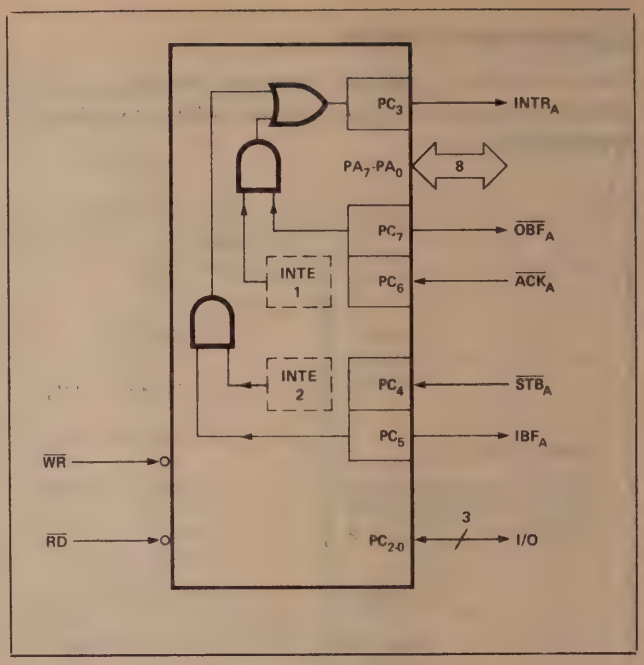


Figure 14. MODE 2

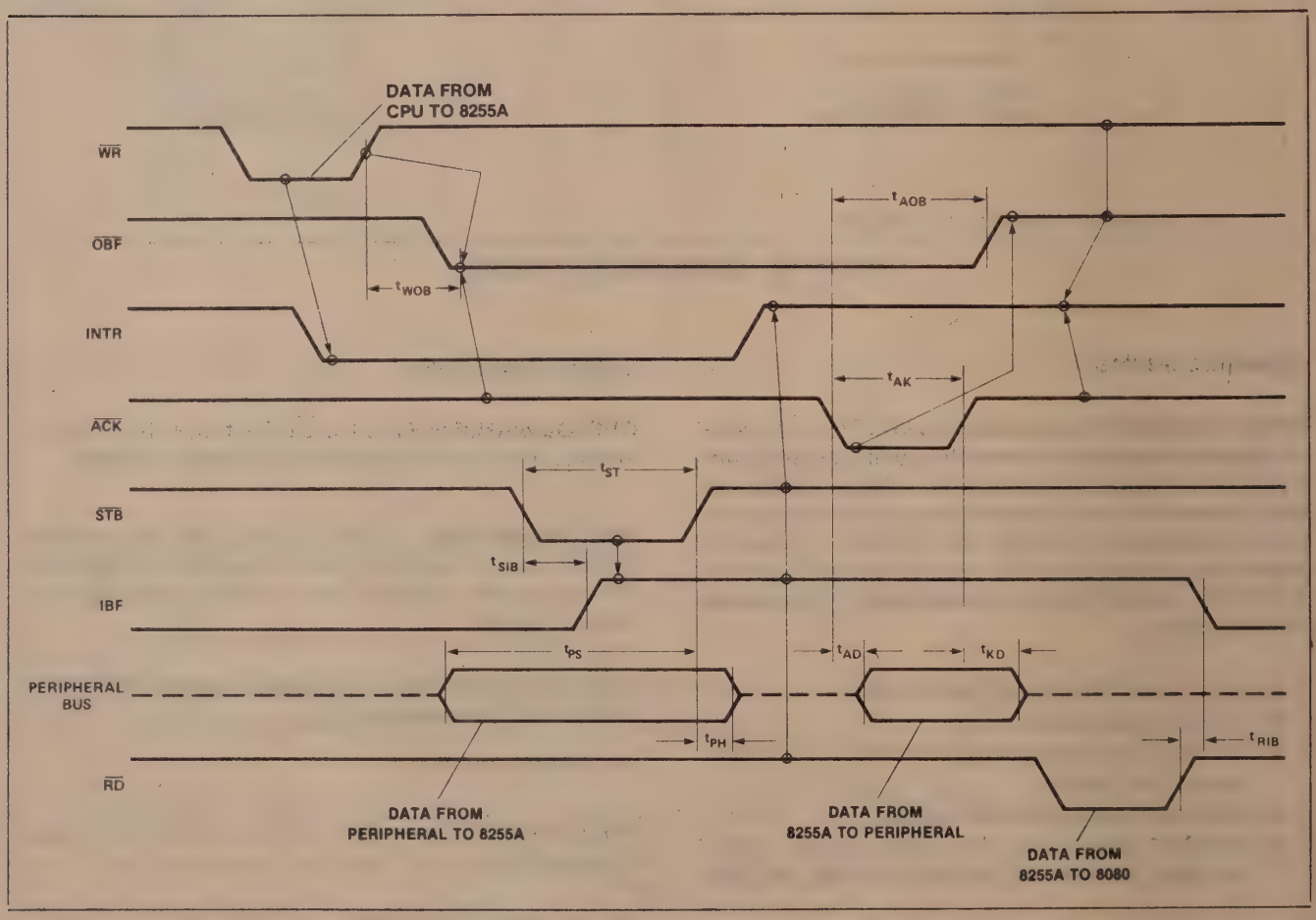


Figure 15. MODE 2 (Bidirectional)

NOTE: Any sequence where $\overline{WR}$ occurs before $\overline{ACK}$ and $\overline{STB}$ occurs before $\overline{RD}$ is permissible.
($INTR = IBF \cdot \overline{MASK} \cdot \overline{STB} \cdot \overline{RD} + \overline{OBF} \cdot \overline{MASK} \cdot \overline{ACK} \cdot \overline{WR}$)

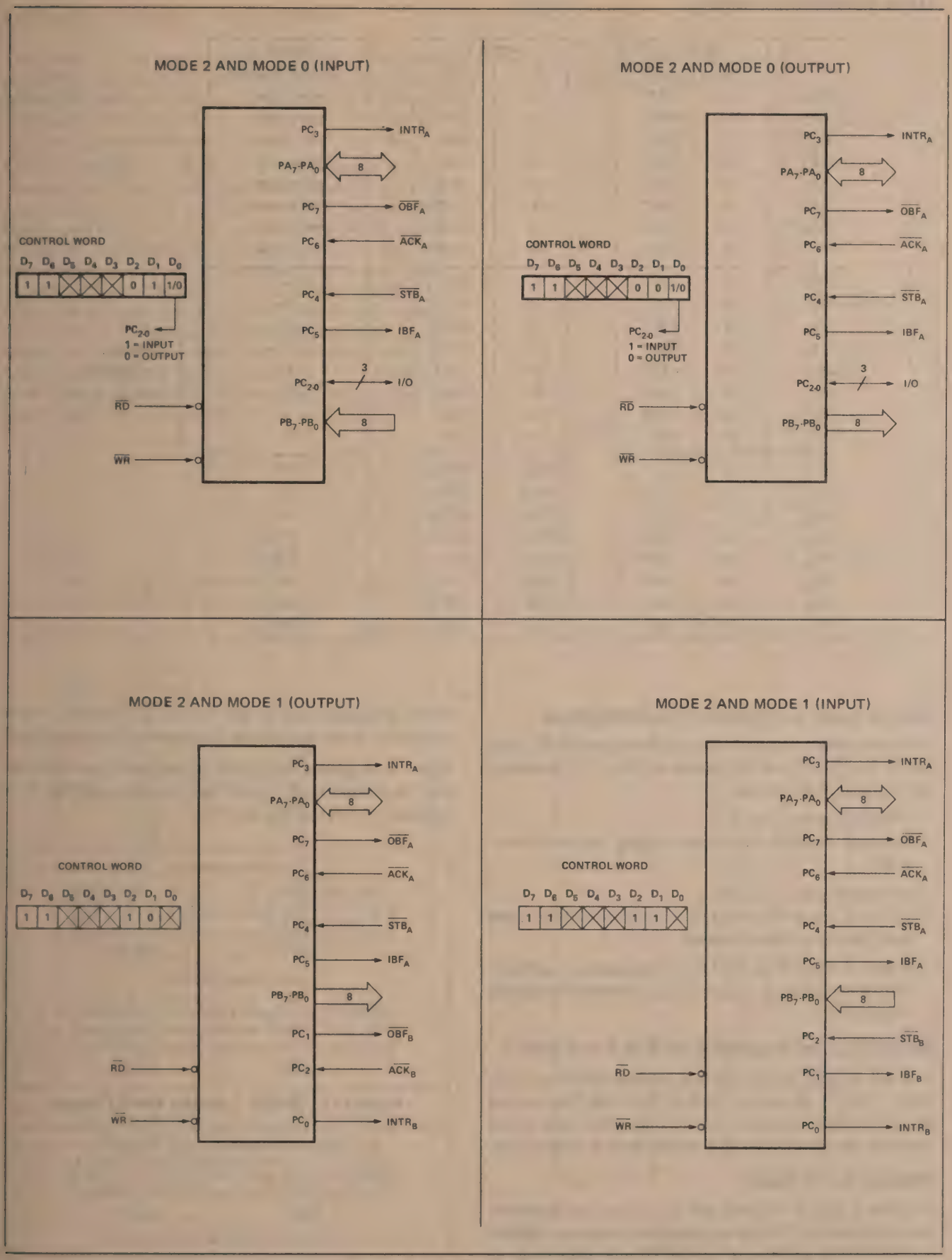


Figure 16. MODE ¼ Combinations

Mode Definition Summary

	MODE 0		MODE 1		MODE 2	
	IN	OUT	IN	OUT	GROUP A ONLY	
PA_0	IN	OUT	IN	OUT	↔	
PA_1	IN	OUT	IN	OUT	↔	
PA_2	IN	OUT	IN	OUT	↔	
PA_3	IN	OUT	IN	OUT	↔	
PA_4	IN	OUT	IN	OUT	↔	
PA_5	IN	OUT	IN	OUT	↔	
PA_6	IN	OUT	IN	OUT	↔	
PA_7	IN	OUT	IN	OUT	↔	
PB_0	IN	OUT	IN	OUT	—	MODE 0 OR MODE 1 ONLY
PB_1	IN	OUT	IN	OUT	—	
PB_2	IN	OUT	IN	OUT	—	
PB_3	IN	OUT	IN	OUT	—	
PB_4	IN	OUT	IN	OUT	—	
PB_5	IN	OUT	IN	OUT	—	
PB_6	IN	OUT	IN	OUT	—	
PB_7	IN	OUT	IN	OUT	—	
PC_0	IN	OUT	$INTR_B$	$INTR_B$	I/O	
PC_1	IN	OUT	IBF_B	$\overline{OBF}_B$	I/O	
PC_2	IN	OUT	$\overline{STB}_B$	$\overline{ACK}_B$	I/O	
PC_3	IN	OUT	$INTR_A$	$INTR_A$	$INTR_A$	
PC_4	IN	OUT	$\overline{STB}_A$	I/O	$\overline{STB}_A$	
PC_5	IN	OUT	IBF_A	I/O	IBF_A	
PC_6	IN	OUT	I/O	$\overline{ACK}_A$	$\overline{ACK}_A$	
PC_7	IN	OUT	I/O	$\overline{OBF}_A$	$\overline{OBF}_A$	

Special Mode Combination Considerations

There are several combinations of modes when not all of the bits in Port C are used for control or status. The remaining bits can be used as follows:

If Programmed as Inputs –
All input lines can be accessed during a normal Port C read.

If Programmed as Outputs –
Bits in C upper (PC_7-PC_4) must be individually accessed using the bit set/reset function.

Bits in C lower (PC_3-PC_0) can be accessed using the bit set/reset function or accessed as a threesome by writing into Port C.

Source Current Capability on Port B and Port C

Any set of eight output buffers, selected randomly from Ports B and C can source 1mA at 1.5 volts. This feature allows the 8255 to directly drive Darlington type drivers and high-voltage displays that require such source current.

Reading Port C Status

In Mode 0, Port C transfers data to or from the peripheral device. When the 8255 is programmed to function in Modes 1 or 2, Port C generates or accepts "hand-shaking" signals with the peripheral device. Reading the contents of Port C allows the programmer to test or verify the "status" of each peripheral device and change the program flow accordingly.

There is no special instruction to read the status information from Port C. A normal read operation of Port C is executed to perform this function.

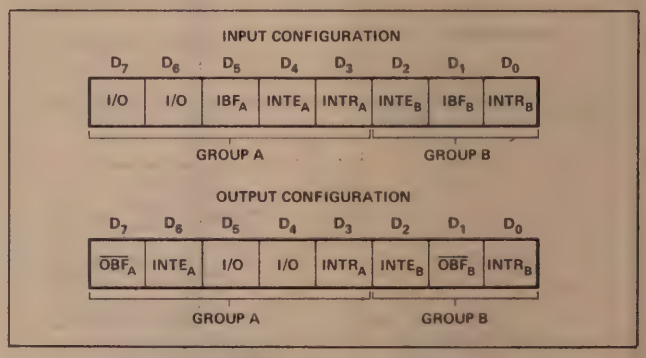


Figure 17. MODE 1 Status Word Format

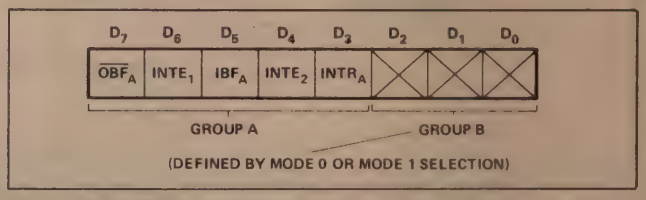


Figure 18. MODE 2 Status Word Format

APPLICATIONS OF THE 8255A

The 8255A is a very powerful tool for interfacing peripheral equipment to the microcomputer system. It represents the optimum use of available pins and is flexible enough to interface almost any I/O device without the need for additional external logic.

Each peripheral device in a microcomputer system usually has a "service routine" associated with it. The routine manages the software interface between the device and the CPU. The functional definition of the 8255A is programmed by the I/O service routine and becomes an extension of the system software. By examining the I/O devices interface characteristics for both data transfer and timing, and matching this information to the examples and tables in the detailed operational description, a control word can easily be developed to initialize the 8255A to exactly "fit" the application. Figures 19 through 25 present a few examples of typical applications of the 8255A.

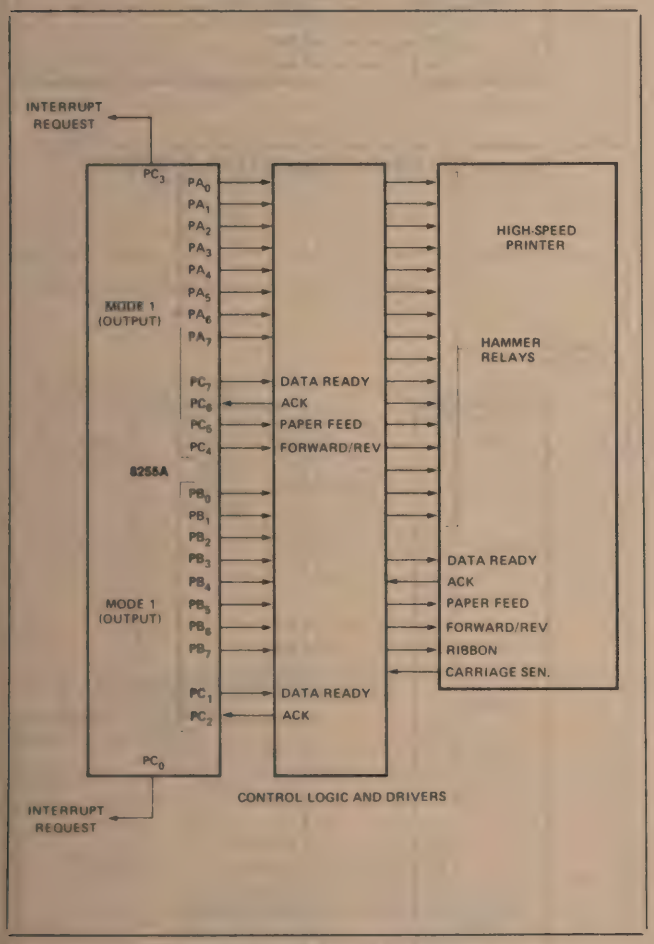


Figure 19. Printer Interface

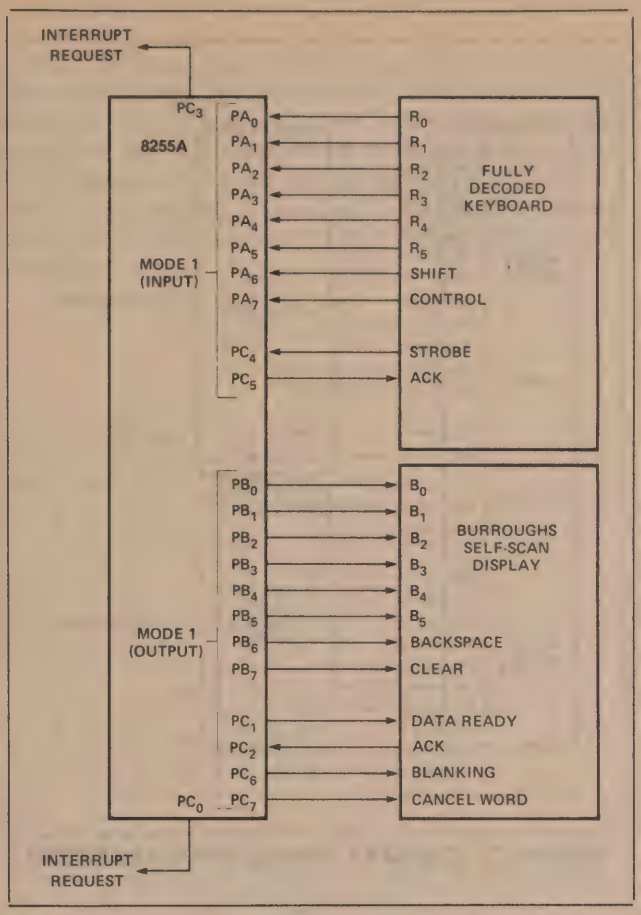


Figure 20. Keyboard and Display Interface

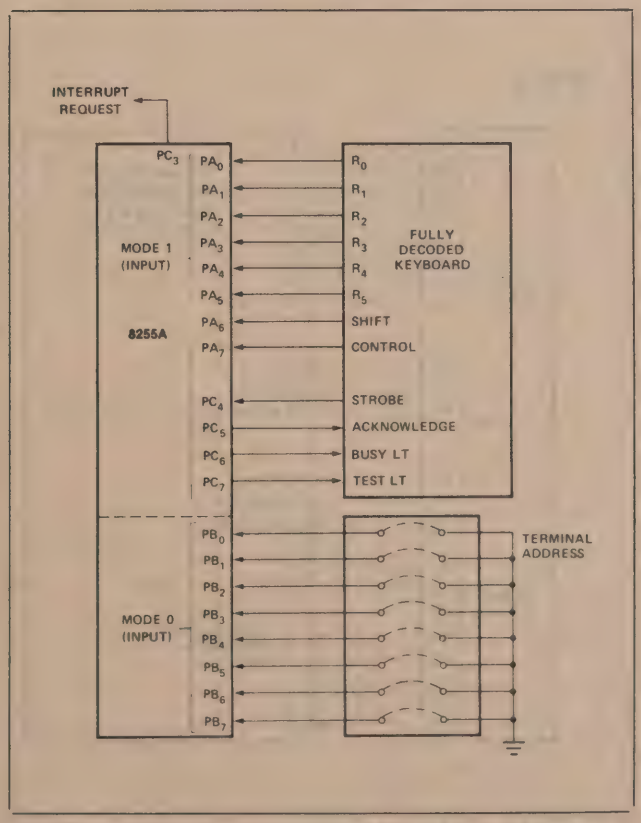


Figure 21. Keyboard and Terminal Address Interface

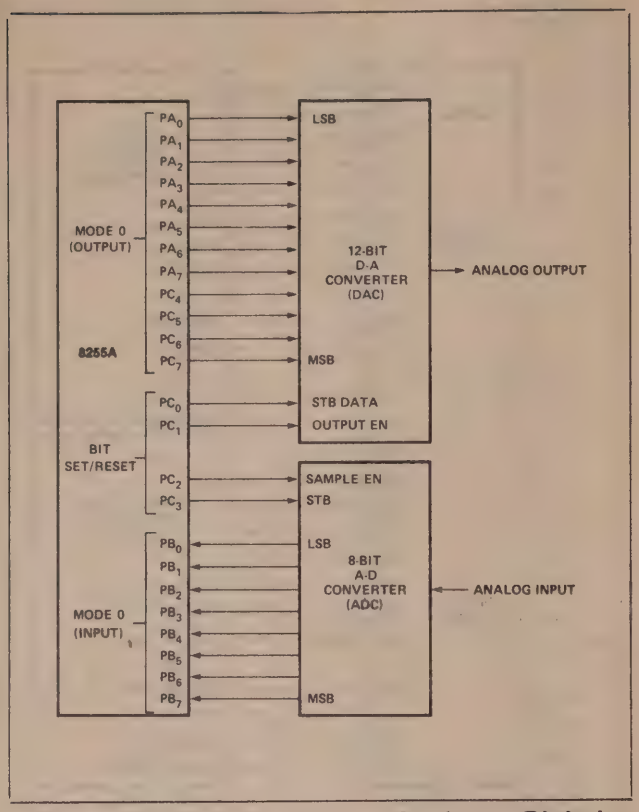


Figure 22. Digital to Analog, Analog to Digital

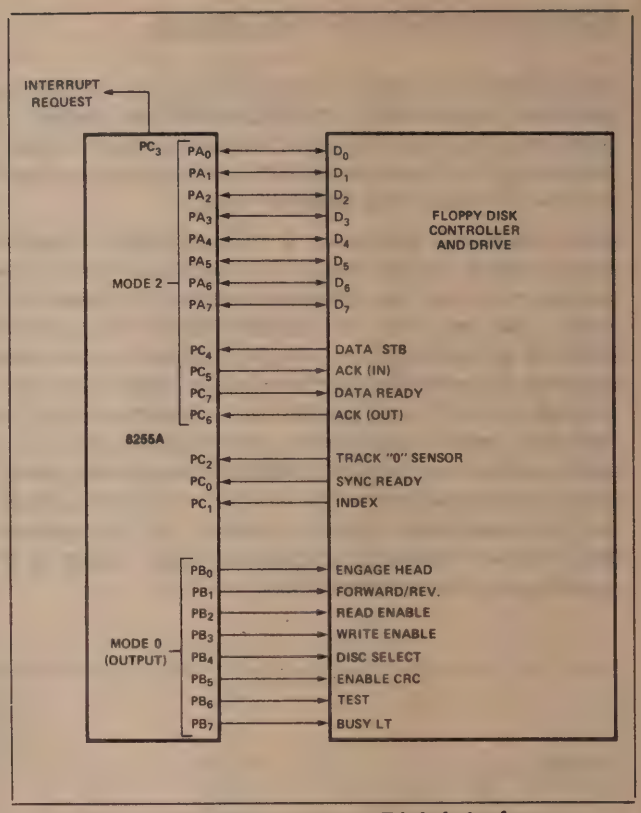


Figure 23. Basic Floppy Disk Interface

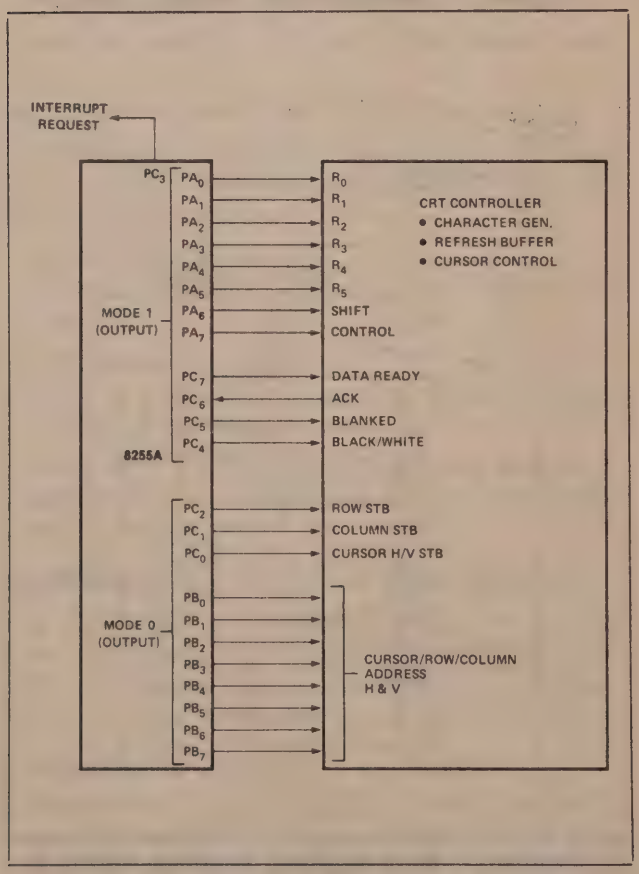


Figure 24. Basic CRT Controller Interface

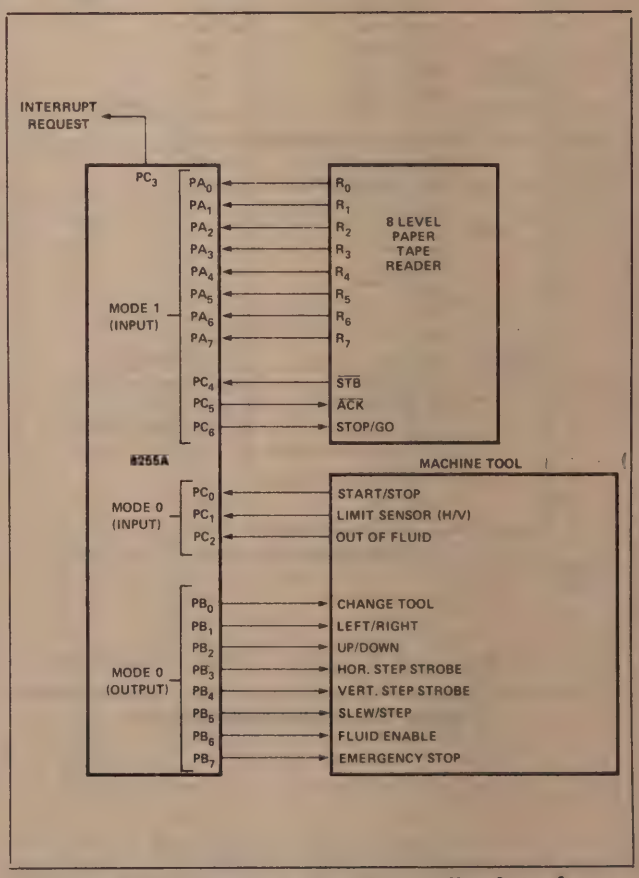


Figure 25. Machine Tool Controller Interface

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias. . . . . 0°C to 70°C
Storage Temperature . . . . . –65°C to +150°C
Voltage on Any Pin
With Respect to Ground . . . . . –0.5V to +7V
Power Dissipation . . . . . 1 Watt

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = +5V ±10%, GND = 0V) *

Symbol	Parameter	Min.	Max.	Unit	Test Conditions
V_{IL}	Input Low Voltage	-0.5	0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC}	V	
V_{OL} (DB)	Output Low Voltage (Data Bus)		0.45*	V	I_{OL} = 2.5mA
V_{OL}(PER)	Output Low Voltage (Peripheral Port)		0.45*	V	I_{OL} = 1.7mA
V_{OH}(DB)	Output High Voltage (Data Bus)	2.4		V	I_{OH} = -400μA
V_{OH}(PER)	Output High Voltage (Peripheral Port)	2.4		V	I_{OH} = -200μA
I_{DAR}[1]	Darlington Drive Current	-1.0	-4.0	mA	R_{EXT} = 750Ω; V_{EXT} = 1.5V
I_{CC}	Power Supply Current		120	mA	
I_{IL}	Input Load Current		±10	μA	V_{IN} = V_{CC} to 0V
I_{OFL}	Output Float Leakage		±10	μA	V_{OUT} = V_{CC} to .45V

NOTE:
1. Available on any 8 pins from Port B and C.

CAPACITANCE (T_A = 25°C, V_{CC} = GND = 0V)

Symbol	Parameter	Min.	Typ.	Max.	Unit	Test Conditions
C_{IN}	Input Capacitance			10	pF	fc = 1MHz
$C_{I/O}$	I/O Capacitance			20	pF	Unmeasured pins returned to GND

A.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = +5V ±10%, GND = 0V) *

Bus Parameters

READ

Symbol	Parameter	8255A		8255A-5		Unit
		Min.	Max.	Min.	Max.	
t_{AR}	Address Stable Before READ	0		0		ns
t_{RA}	Address Stable After READ	0		0		ns
t_{RR}	READ Pulse Width	300		300		ns
t_{RD}	Data Valid From READ[1]		250		200	ns
t_{DF}	Data Float After READ	10	150	10	100	ns
t_{RV}	Time Between READs and/or WRITEs	850		850		ns

A.C. CHARACTERISTICS (Continued)

WRITE

Symbol	Parameter	8255A Min.	8255A Max.	8255A-5 Min.	8255A-5 Max.	Unit
t_{AW}	Address Stable Before WRITE	0		0		ns
t_{WA}	Address Stable After WRITE	20		20		ns
t_{WW}	WRITE Pulse Width	400		300		ns
t_{DW}	Data Valid to WRITE (T.E.)	100		100		ns
t_{WD}	Data Valid After WRITE	30		30		ns

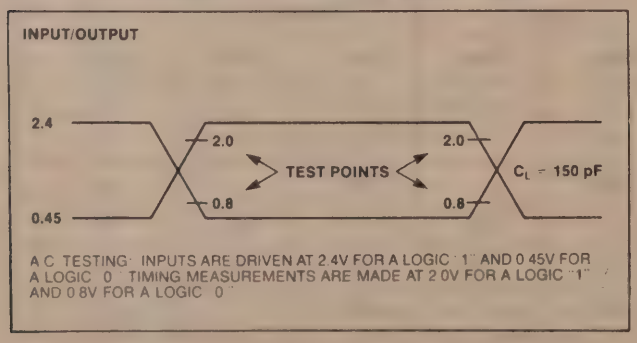
OTHER TIMINGS

Symbol	Parameter	8255A Min.	8255A Max.	8255A-5 Min.	8255A-5 Max.	Unit
t_{WB}	WR = 1 to Output[1]		350		350	ns
t_{IR}	Peripheral Data Before RD	0		0		ns
t_{HR}	Peripheral Data After RD	0		0		ns
t_{AK}	ACK Pulse Width	300		300		ns
t_{ST}	STB Pulse Width	500		500		ns
t_{PS}	Per. Data Before T.E. of STB	0		0		ns
t_{PH}	Per. Data After T.E. of STB	180		180		ns
t_{AD}	ACK = 0 to Output[1]		300		300	ns
t_{KD}	ACK = 1 to Output Float	20	250	20	250	ns
t_{WOB}	WR = 1 to OBF = 0[1]		650		650	ns
t_{AOB}	ACK = 0 to OBF = 1[1]		350		350	ns
t_{SIB}	STB = 0 to IBF = 1[1]		300		300	ns
t_{RIB}	RD = 1 to IBF = 0[1]		300		300	ns
t_{RIT}	RD = 0 to INTR = 0[1]		400		400	ns
t_{SIT}	STB = 1 to INTR = 1[1]		300		300	ns
t_{AIT}	ACK = 1 to INTR = 1[1]		350		350	ns
t_{WIT}	WR = 0 to INTR = 0[1,3]		450		450	ns

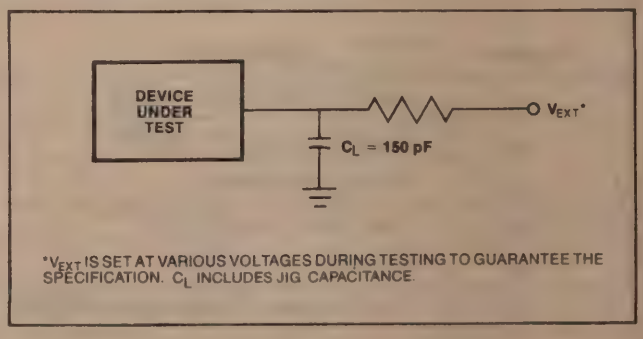
NOTES:
1. Test Conditions: C_L = 150 pF.
2. Period of Reset pulse must be at least 50μs during or after power on. Subsequent Reset pulse can be 500 ns min.
3. INTR↑ may occur as early as $\overline{WR}$↓.

* For Extended Temperature EXPRESS, use M8255A electrical parameters.

A.C. TESTING INPUT, OUTPUT WAVEFORM

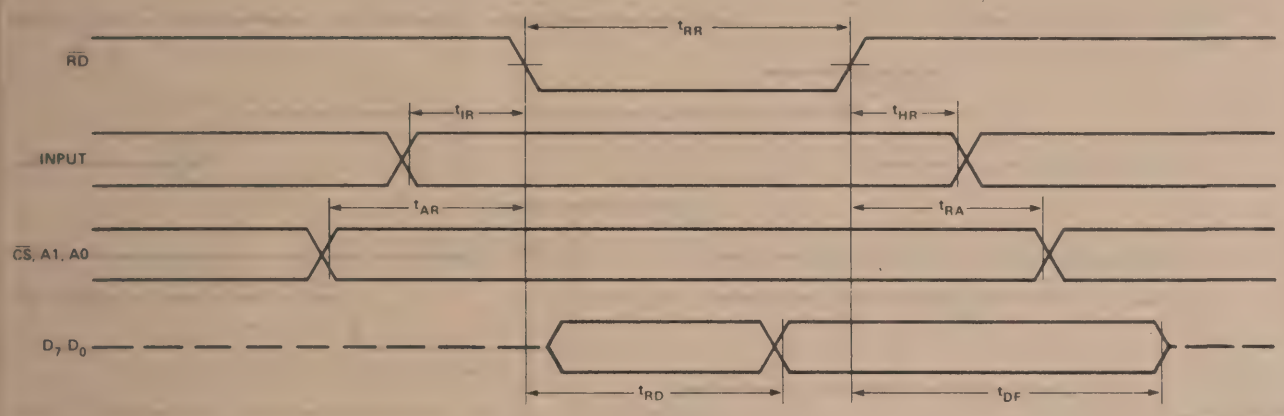


A.C. TESTING LOAD CIRCUIT

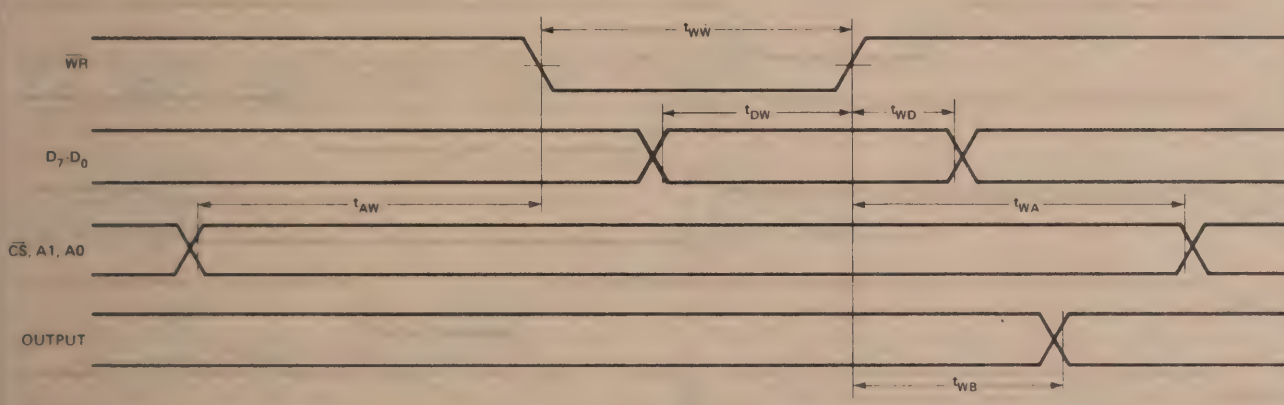


WAVEFORMS

MODE 0 (BASIC INPUT)

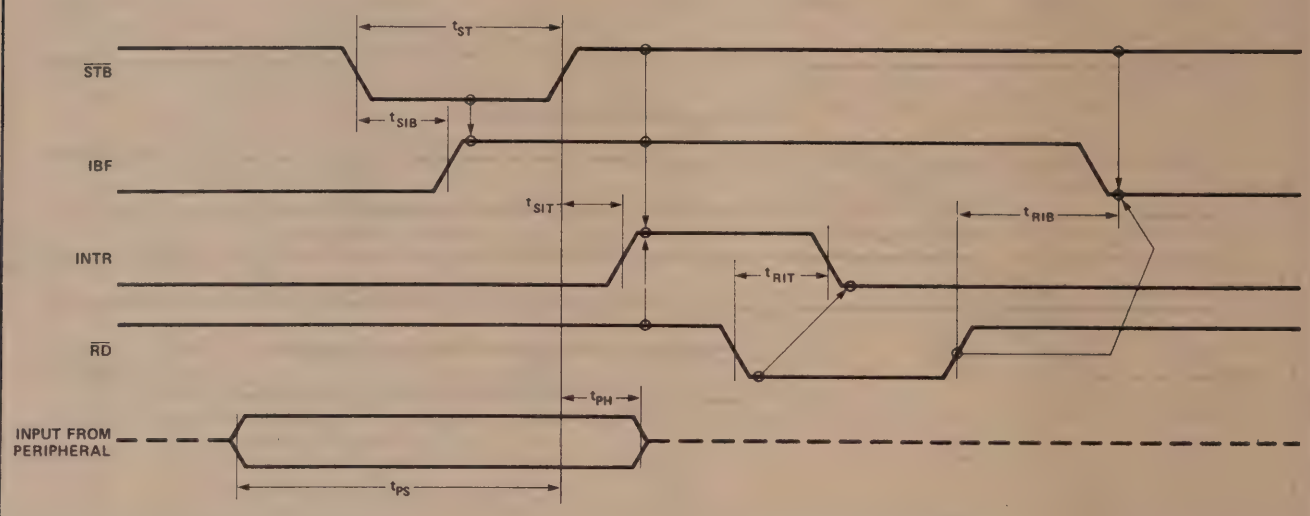


MODE 0 (BASIC OUTPUT)

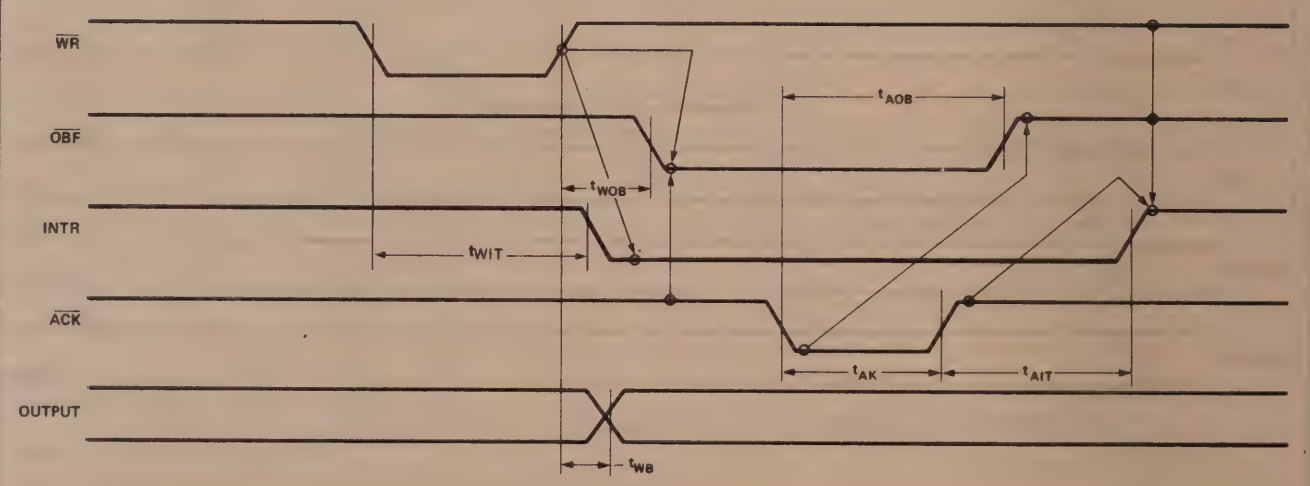


WAVEFORMS (Continued)

MODE 1 (STROBED INPUT)

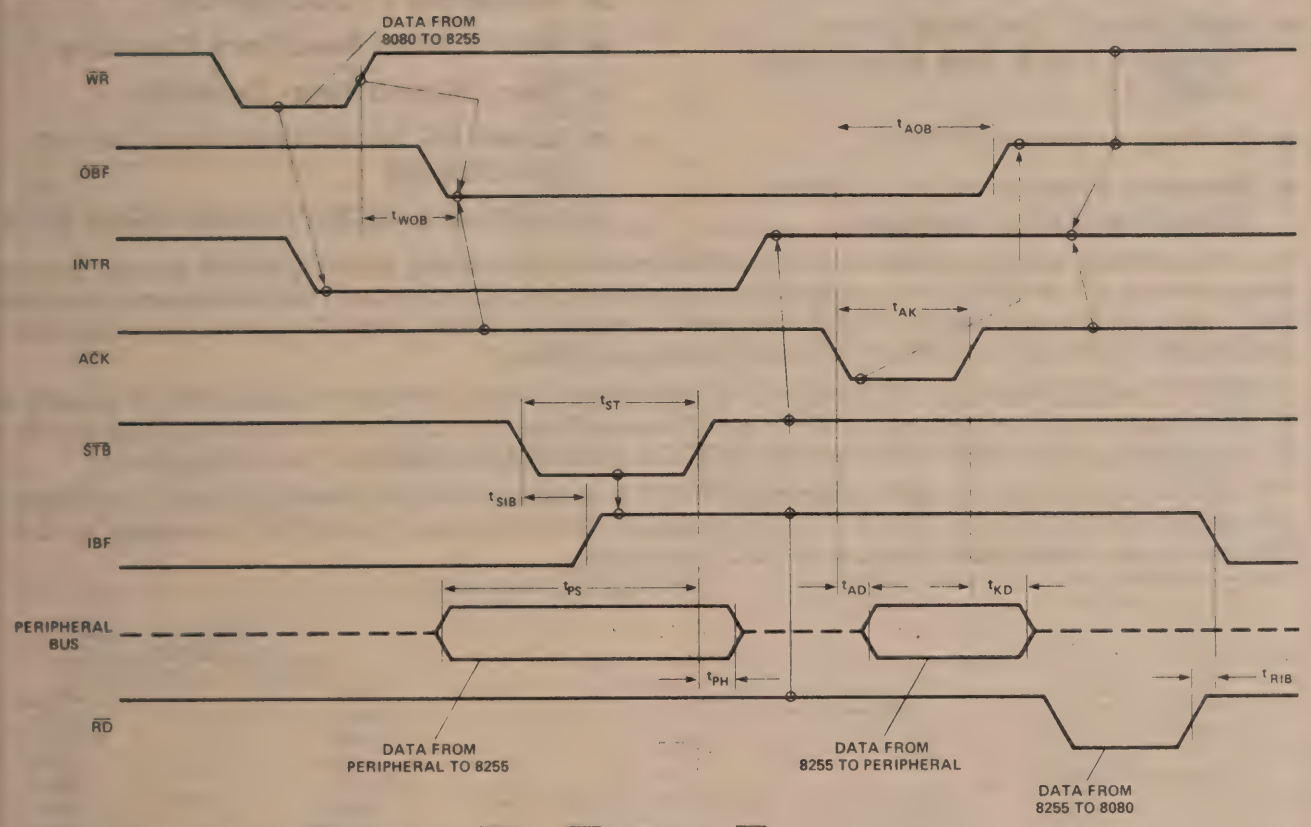


MODE 1 (STROBED OUTPUT)



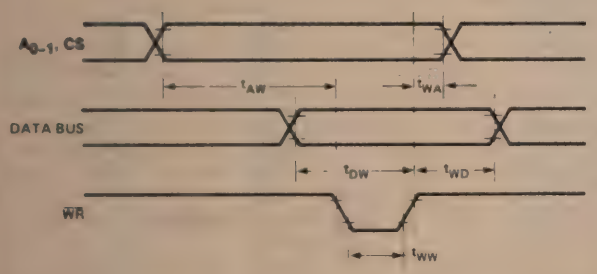
WAVEFORMS (Continued)

MODE 2 (BIDIRECTIONAL)

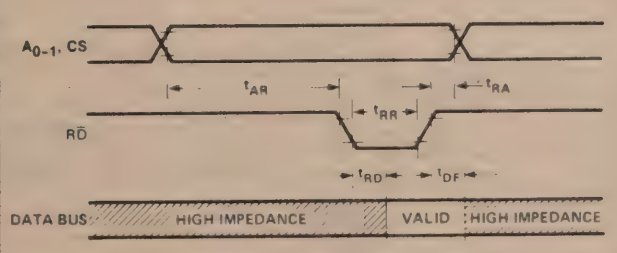


NOTE: Any sequence where $\overline{WR}$ occurs before $\overline{ACK}$ and $\overline{STB}$ occurs before $\overline{RD}$ is permissible.
(INTR = IBF • $\overline{MASK}$ • $\overline{STB}$ • $\overline{RD}$ + $\overline{OBF}$ • $\overline{MASK}$ • $\overline{ACK}$ • $\overline{WR}$)

WRITE TIMING



READ TIMING



82C55A FUNCTIONAL DESCRIPTION

General

The 82C55A is a programmable peripheral interface device designed for use in Intel microcomputer systems. Its function is that of a general purpose I/O component to interface peripheral equipment to the microcomputer system bus. The functional configuration of the 82C55A is programmed by the system software so that normally no external logic is necessary to interface peripheral devices or structures.

Data Bus Buffer

This 3-state bidirectional 8-bit buffer is used to interface the 82C55A to the system data bus. Data is transmitted or received by the buffer upon execution of input or output instructions by the CPU. Control words and status information are also transferred through the data bus buffer.

Read/Write and Control Logic

The function of this block is to manage all of the internal and external transfers of both Data and Control or Status words. It accepts inputs from the CPU Address and Control busses and in turn, issues commands to both of the Control Groups.

Group A and Group B Controls

The functional configuration of each port is programmed by the systems software. In essence, the CPU "outputs" a control word to the 82C55A. The control word contains information such as "mode", "bit set", "bit reset", etc., that initializes the functional configuration of the 82C55A.

Each of the Control blocks (Group A and Group B) accepts "commands" from the Read/Write Control Logic, receives "control words" from the internal data bus and issues the proper commands to its associated ports.

Control Group A - Port A and Port C upper (C7–C4)
Control Group B - Port B and Port C lower (C3–C0)

The control word register can be both written and read as shown in the address decode table in the pin descriptions. Figure 6 shows the control word format for both Read and Write operations. When the control word is read, bit D7 will always be a logic "1", as this implies control word mode information.

Ports A, B, and C

The 82C55A contains three 8-bit ports (A, B, and C). All can be configured in a wide variety of functional characteristics by the system software but each has its own special features or "personality" to further enhance the power and flexibility of the 82C55A.

Port A. One 8-bit data output latch/buffer and one 8-bit input latch buffer. Both "pull-up" and "pull-down" bus hold devices are present on Port A.

Port B. One 8-bit data input/output latch/buffer. Only "pull-up" bus hold devices are present on Port B.

Port C. One 8-bit data output latch/buffer and one 8-bit data input buffer (no latch for input). This port can be divided into two 4-bit ports under the mode control. Each 4-bit port contains a 4-bit latch and it can be used for the control signal outputs and status signal inputs in conjunction with ports A and B. Only "pull-up" bus hold devices are present on Port C.

See Figure 4 for the bus-hold circuit configuration for Port A, B, and C.

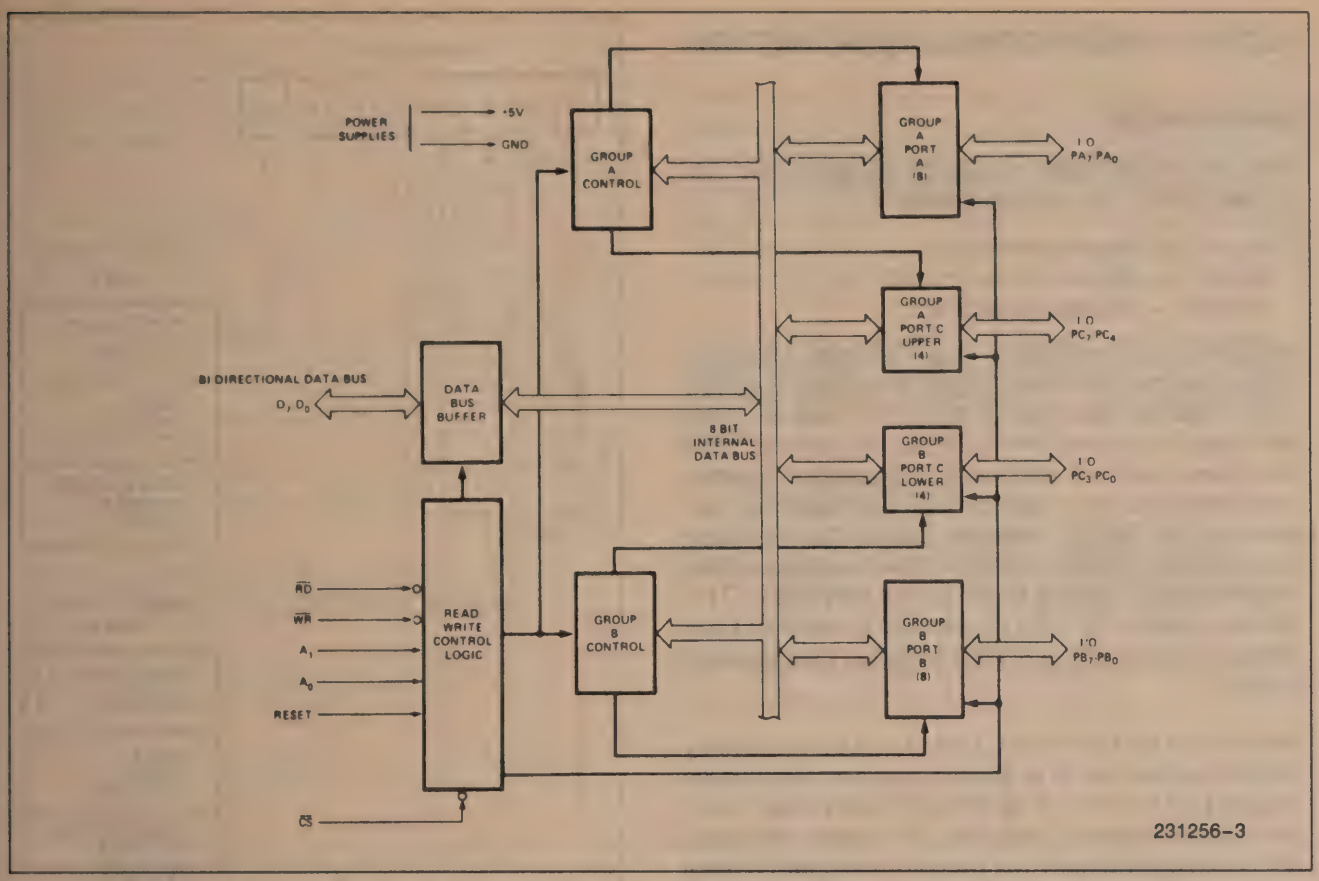


Figure 3. 82C55A Block Diagram Showing Data Bus Buffer and Read/Write Control Logic Functions

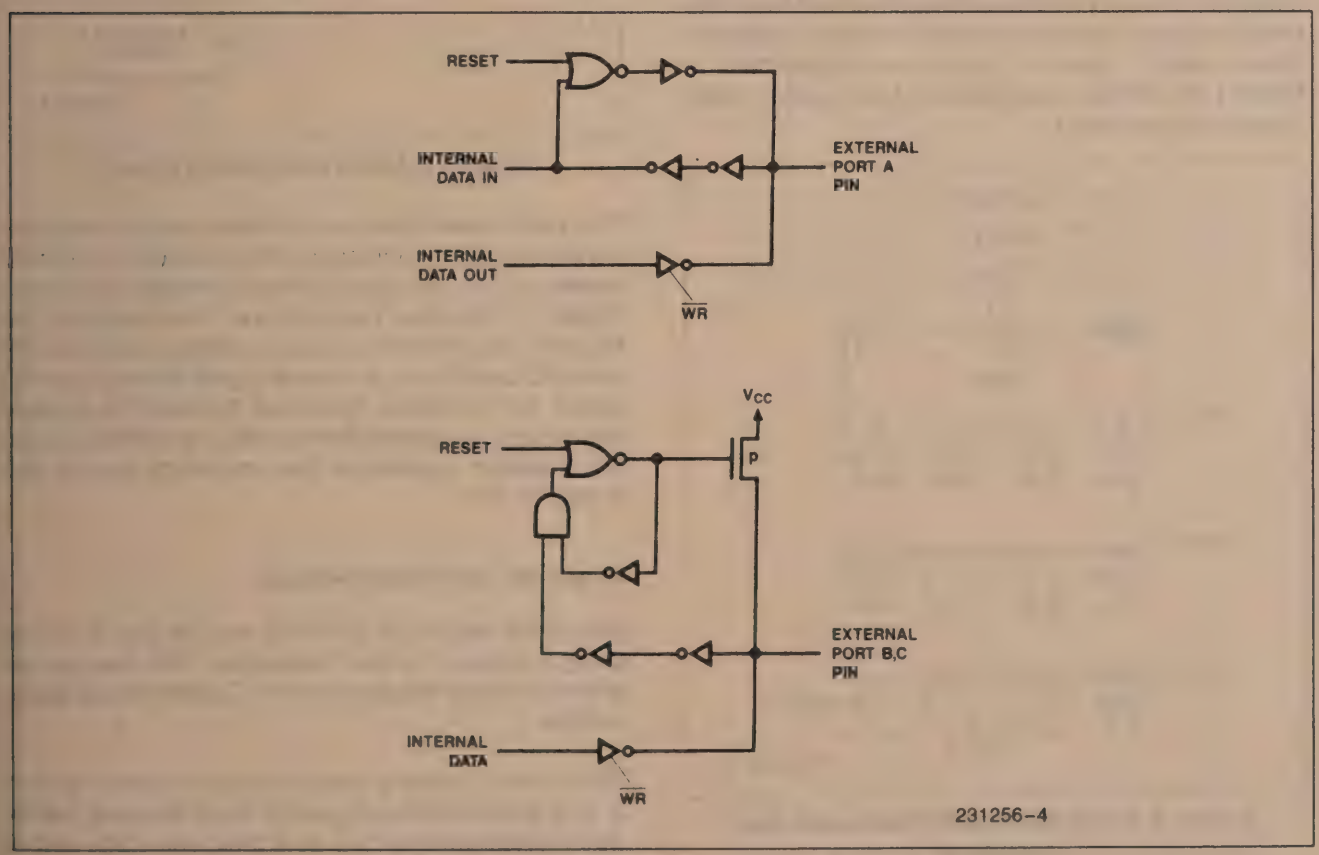


Figure 4. Port A, B, C, Bus-hold Configuration

82C55A OPERATIONAL DESCRIPTION

Mode Selection

There are three basic modes of operation that can be selected by the system software:

Mode 0 — Basic input/output
Mode 1 — Strobed Input/output
Mode 2 — Bi-directional Bus

When the reset input goes "high" all ports will be set to the input mode with all 24 port lines held at a logic "one" level by the internal bus hold devices. After the reset is removed the 82C55A can remain in the input mode with no additional initialization required. This eliminates the need for pullup or pulldown devices in "all CMOS" designs. During the execution of the system program, any of the other modes may be selected by using a single output instruction. This allows a single 82C55A to service a variety of peripheral devices with a simple software maintenance routine.

The modes for Port A and Port B can be separately defined, while Port C is divided into two portions as required by the Port A and Port B definitions. All of the output registers, including the status flip-flops, will be reset whenever the mode is changed. Modes may be combined so that their functional definition can be "tailored" to almost any I/O structure. For instance; Group B can be programmed in Mode 0 to monitor simple switch closings or display computational results, Group A could be programmed in Mode 1 to monitor a keyboard or tape reader on an interrupt-driven basis.

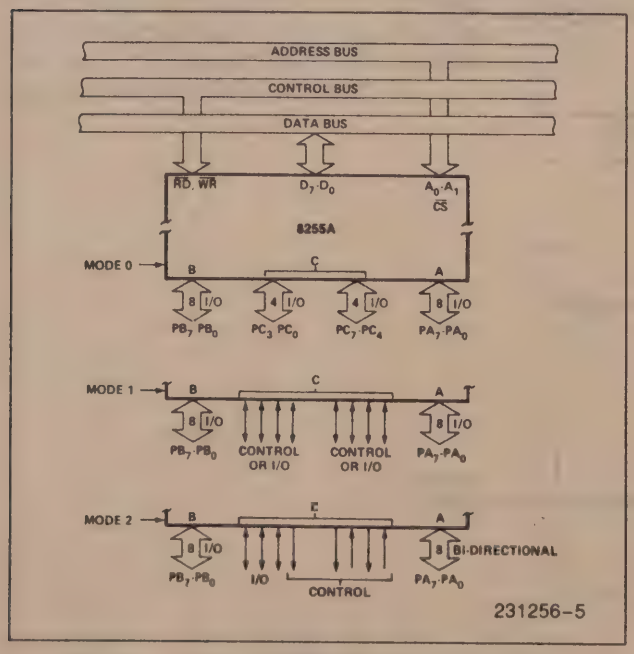


Figure 5. Basic Mode Definitions and Bus Interface

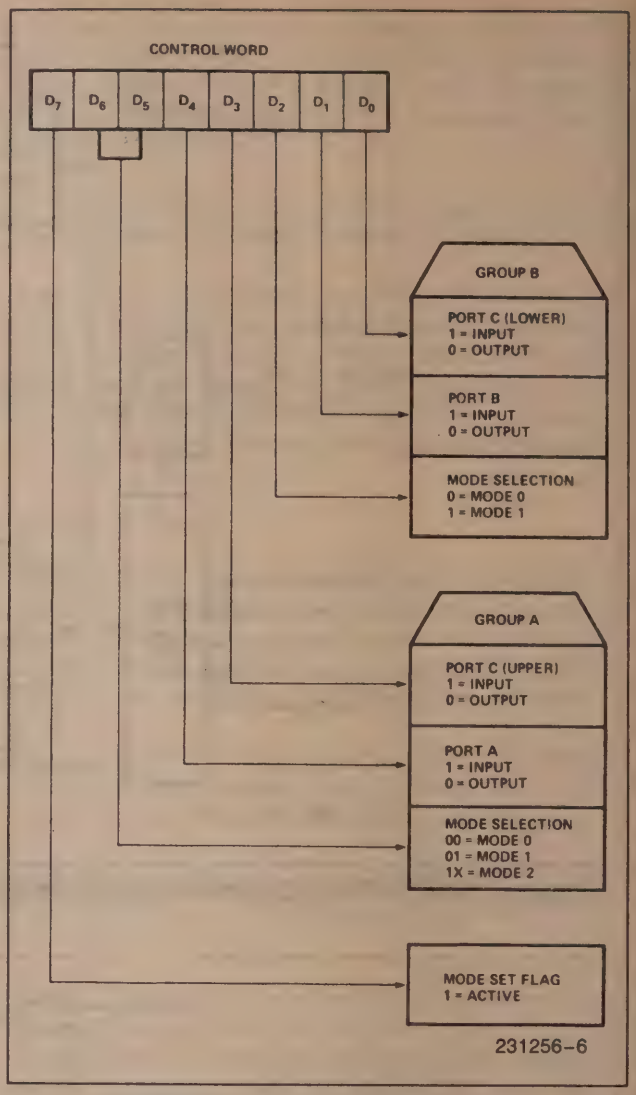


Figure 6. Mode Definition Format

The mode definitions and possible mode combinations may seem confusing at first but after a cursory review of the complete device operation a simple, logical I/O approach will surface. The design of the 82C55A has taken into account things such as efficient PC board layout, control signal definition vs PC layout and complete functional flexibility to support almost any peripheral device with no external logic. Such design represents the maximum use of the available pins.

Single Bit Set/Reset Feature

Any of the eight bits of Port C can be Set or Reset using a single OUTput instruction. This feature reduces software requirements in Control-based applications.

When Port C is being used as status/control for Port A or B, these bits can be set or reset by using the Bit Set/Reset operation just as if they were data output ports.

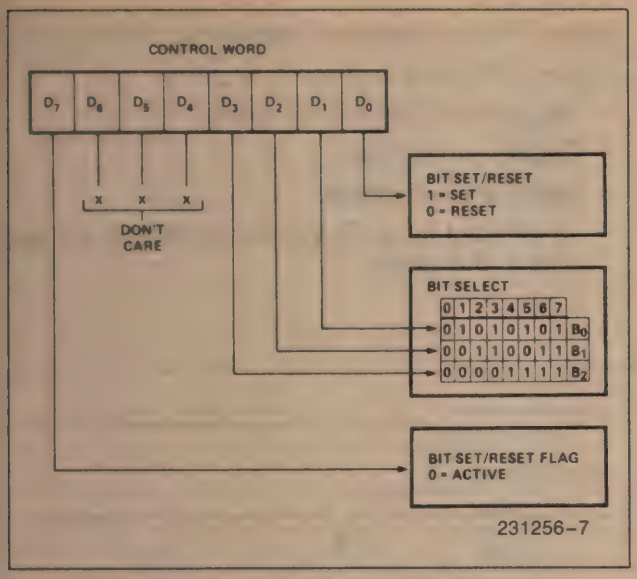


Figure 7. Bit Set/Reset Format

Interrupt Control Functions

When the 82C55A is programmed to operate in mode 1 or mode 2, control signals are provided that can be used as interrupt request inputs to the CPU. The interrupt request signals, generated from port C, can be inhibited or enabled by setting or resetting the associated INTE flip-flop, using the bit set/reset function of port C.

This function allows the Programmer to disallow or allow a specific I/O device to interrupt the CPU without affecting any other device in the interrupt structure.

INTE flip-flop definition:

(BIT-SET)—INTE is SET—Interrupt enable
(BIT-RESET)—INTE is RESET—Interrupt disable

Note:
All Mask flip-flops are automatically reset during mode selection and device Reset.

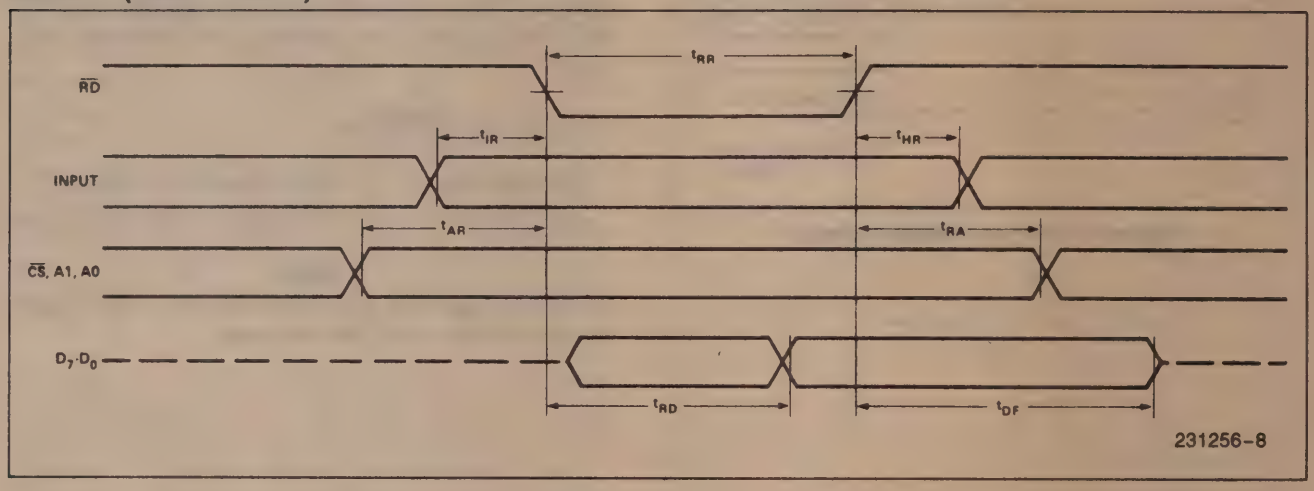
Operating Modes

Mode 0 (Basic Input/Output). This functional configuration provides simple input and output operations for each of the three ports. No "handshaking" is required, data is simply written to or read from a specified port.

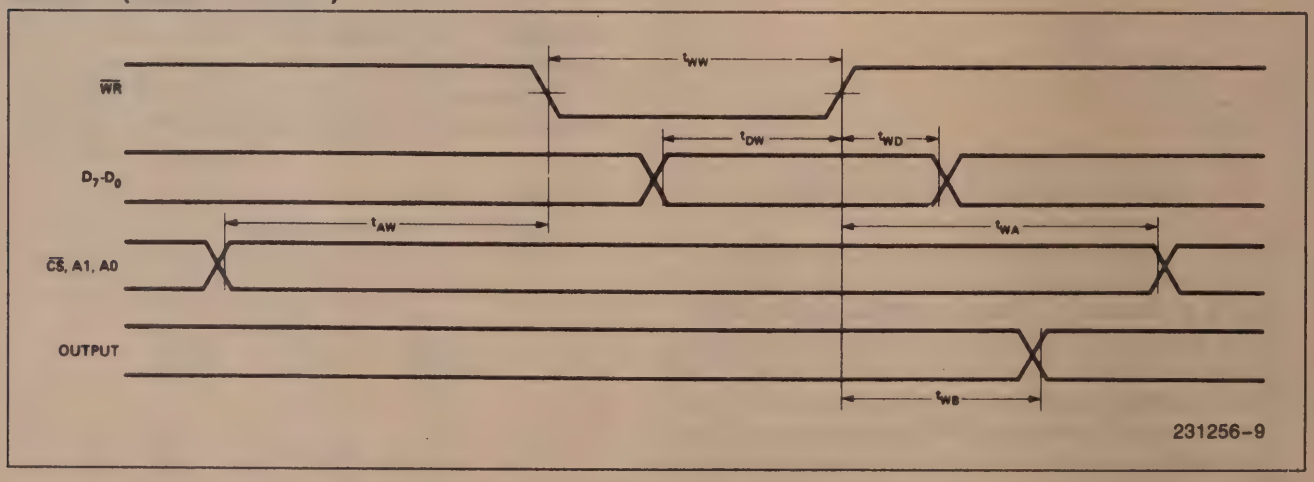
Mode 0 Basic Functional Definitions:

- Two 8-bit ports and two 4-bit ports.
- Any port can be input or output.
- Outputs are latched.
- Inputs are not latched.
- 16 different Input/Output configurations are possible in this Mode.

MODE 0 (BASIC INPUT)



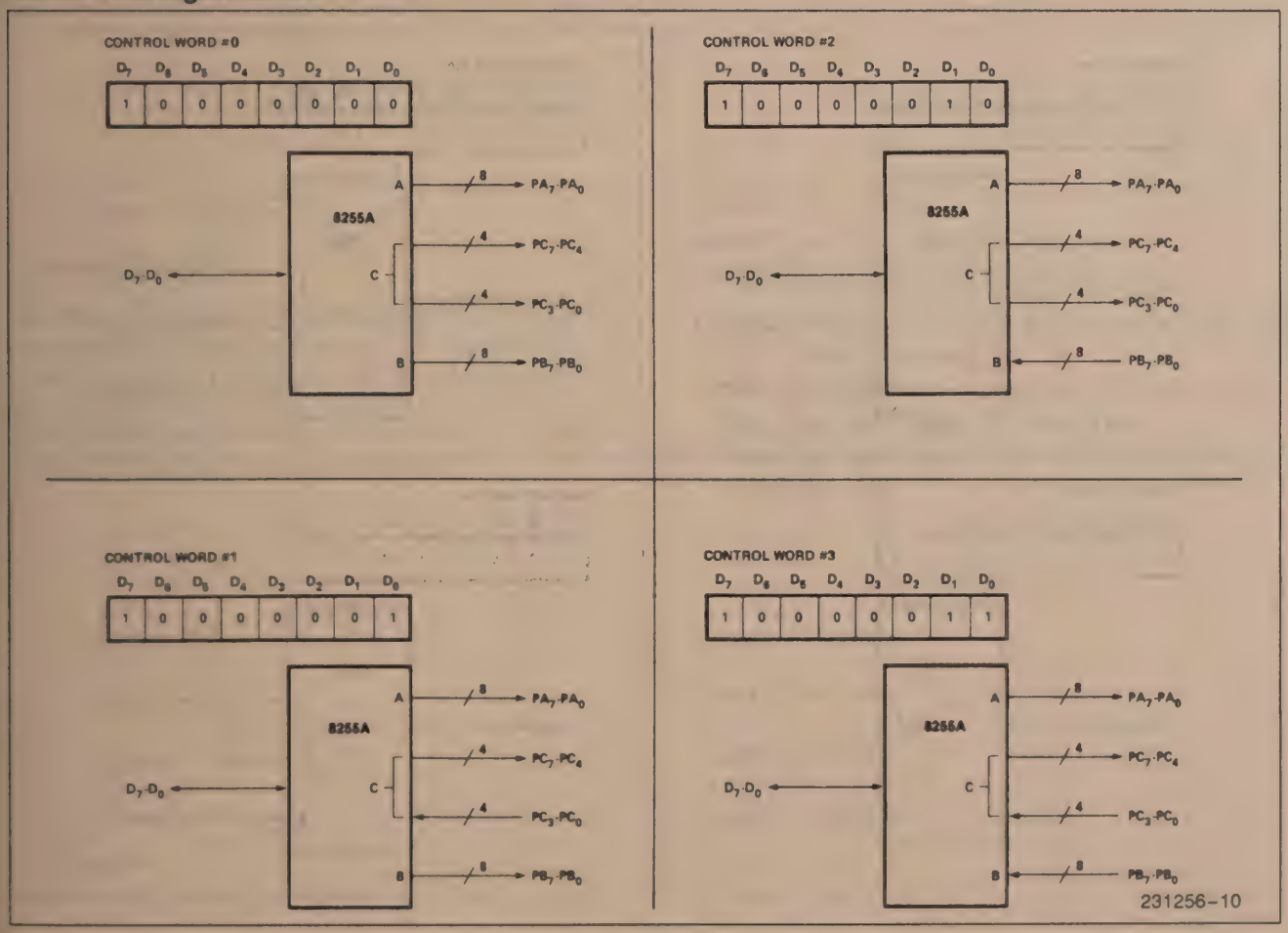
MODE 0 (BASIC OUTPUT)



MODE 0 Port Definition

A		B		GROUP A			GROUP B	
D_4	D_3	D_1	D_0	PORT A	PORT C (UPPER)	#	PORT B	PORT C (LOWER)
0	0	0	0	OUTPUT	OUTPUT	0	OUTPUT	OUTPUT
0	0	0	1	OUTPUT	OUTPUT	1	OUTPUT	INPUT
0	0	1	0	OUTPUT	OUTPUT	2	INPUT	OUTPUT
0	0	1	1	OUTPUT	OUTPUT	3	INPUT	INPUT
0	1	0	0	OUTPUT	INPUT	4	OUTPUT	OUTPUT
0	1	0	1	OUTPUT	INPUT	5	OUTPUT	INPUT
0	1	1	0	OUTPUT	INPUT	6	INPUT	OUTPUT
0	1	1	1	OUTPUT	INPUT	7	INPUT	INPUT
1	0	0	0	INPUT	OUTPUT	8	OUTPUT	OUTPUT
1	0	0	1	INPUT	OUTPUT	9	OUTPUT	INPUT
1	0	1	0	INPUT	OUTPUT	10	INPUT	OUTPUT
1	0	1	1	INPUT	OUTPUT	11	INPUT	INPUT
1	1	0	0	INPUT	INPUT	12	OUTPUT	OUTPUT
1	1	0	1	INPUT	INPUT	13	OUTPUT	INPUT
1	1	1	0	INPUT	INPUT	14	INPUT	OUTPUT
1	1	1	1	INPUT	INPUT	15	INPUT	INPUT

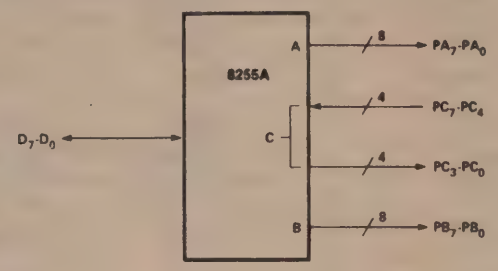
MODE 0 Configurations



MODE 0 Configurations (Continued)

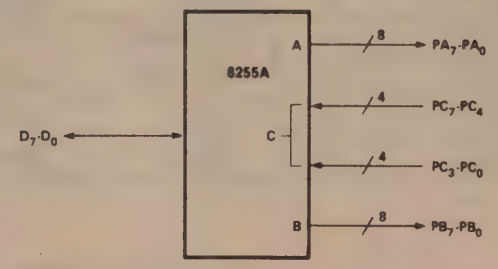
CONTROL WORD #4

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	1	0	0	0



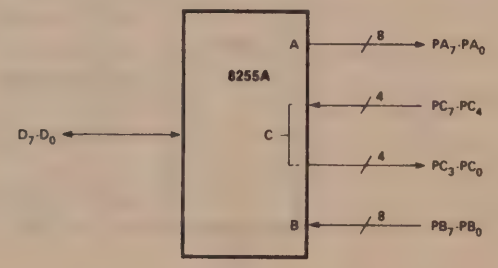
CONTROL WORD #5

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	1	0	0	1



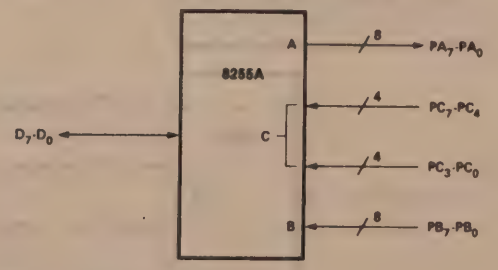
CONTROL WORD #6

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	1	0	1	0



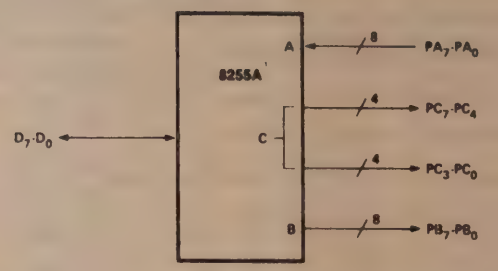
CONTROL WORD #7

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	1	0	1	1



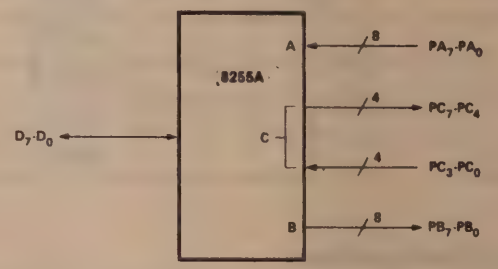
CONTROL WORD #8

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	0	0	0	0



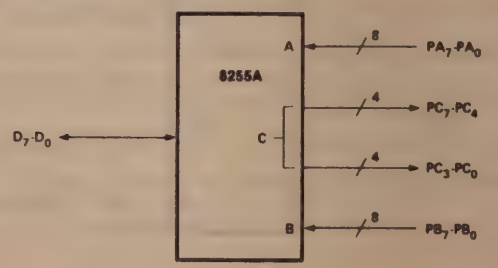
CONTROL WORD #9

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	0	0	0	1



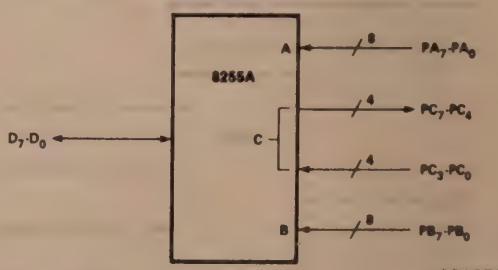
CONTROL WORD #10

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	0	0	1	0



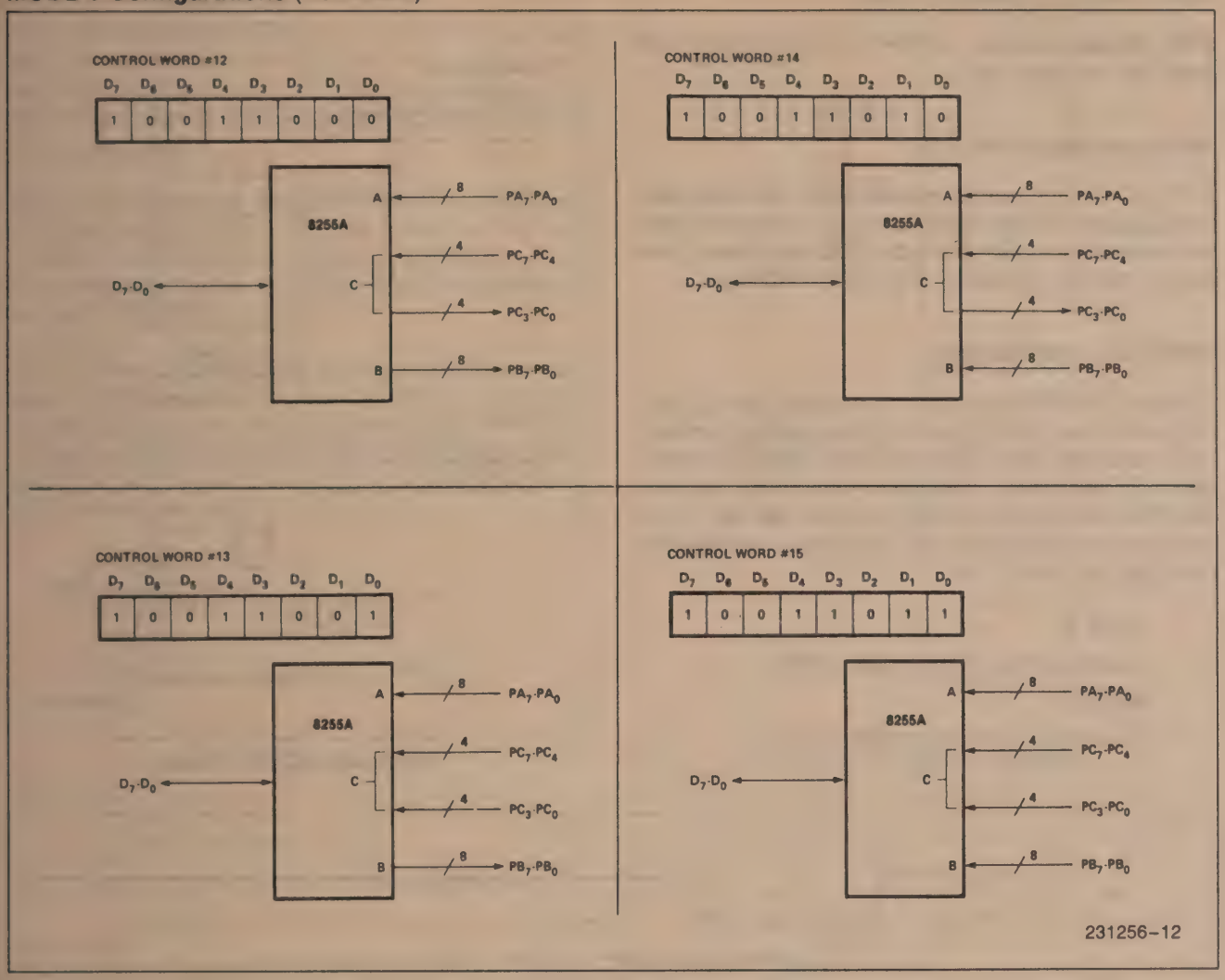
CONTROL WORD #11

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	0	0	1	1



231256-11

MODE 0 Configurations (Continued)



Operating Modes

MODE 1 (Strobed Input/Output). This functional configuration provides a means for transferring I/O data to or from a specified port in conjunction with strobes or "handshaking" signals. In mode 1, Port A and Port B use the lines on Port C to generate or accept these "handshaking" signals.

Mode 1 Basic functional Definitions:

- Two Groups (Group A and Group B).
- Each group contains one 8-bit data port and one 4-bit control/data port.
- The 8-bit data port can be either input or output Both inputs and outputs are latched.
- The 4-bit port is used for control and status of the 8-bit data port.

Input Control Signal Definition

$\overline{STB}$ (Strobe Input). A "low" on this input loads data into the input latch.

IBF (Input Buffer Full F/F)

A "high" on this output indicates that the data has been loaded into the input latch; in essence, an acknowledgement. IBF is set by $\overline{STB}$ input being low and is reset by the rising edge of the $\overline{RD}$ input.

INTR (Interrupt Request)

A "high" on this output can be used to interrupt the CPU when an input device is requesting service. INTR is set by the $\overline{STB}$ is a "one", IBF is a "one" and INTE is a "one". It is reset by the falling edge of $\overline{RD}$. This procedure allows an input device to request service from the CPU by simply strobing its data into the port.

INTE A

Controlled by bit set/reset of PC_4.

INTE B

Controlled by bit set/reset of PC_2.

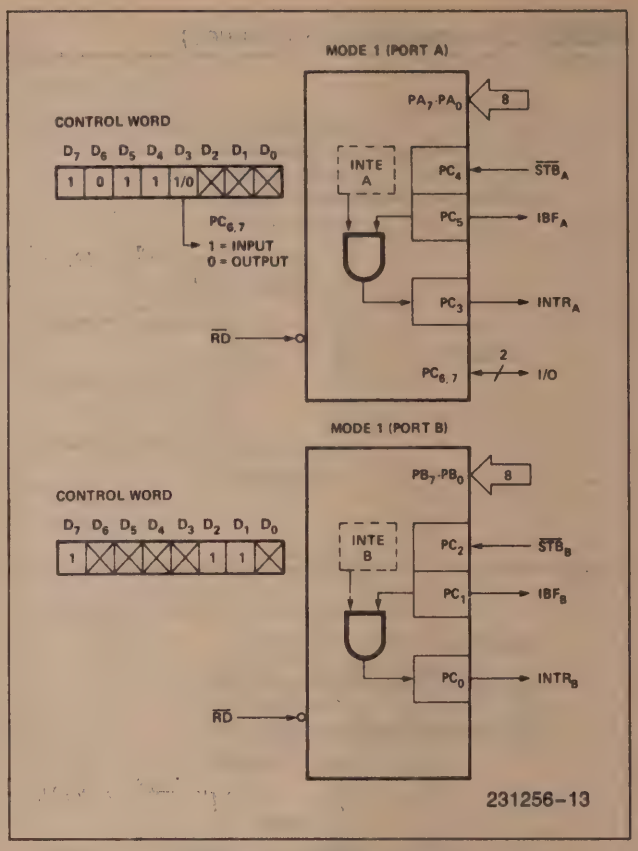


Figure 8. MODE 1 Input

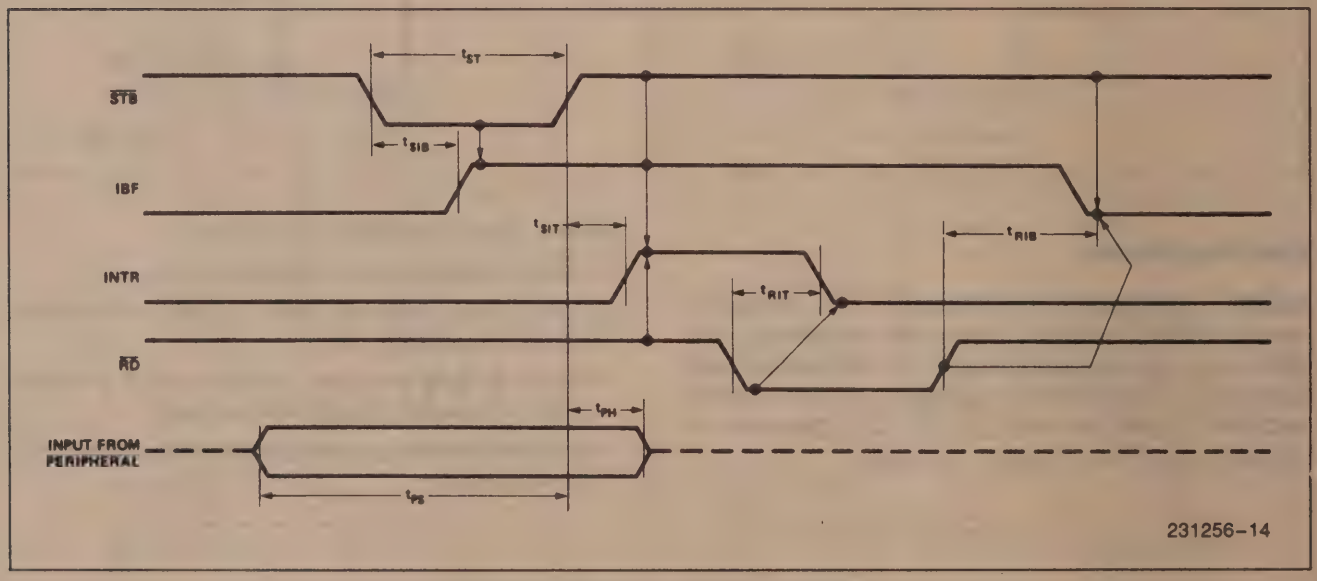


Figure 9. MODE 1 (Strobed Input)

Output Control Signal Definition

$\overline{OBF}$ (Output Buffer Full F/F). The $\overline{OBF}$ output will go "low" to indicate that the CPU has written data out to the specified port. The $\overline{OBF}$ F/F will be set by the rising edge of the $\overline{WR}$ input and reset by $\overline{ACK}$ Input being low.

$\overline{ACK}$ (Acknowledge Input). A "low" on this input informs the 82C55A that the data from Port A or Port B has been accepted. In essence, a response from the peripheral device indicating that it has received the data output by the CPU.

INTR (Interrupt Request). A "high" on this output can be used to interrupt the CPU when an output device has accepted data transmitted by the CPU. INTR is set when $\overline{ACK}$ is a "one", $\overline{OBF}$ is a "one" and INTE is a "one". It is reset by the falling edge of $\overline{WR}$.

INTE A

Controlled by bit set/reset of PC_6.

INTE B

Controlled by bit set/reset of PC_2.

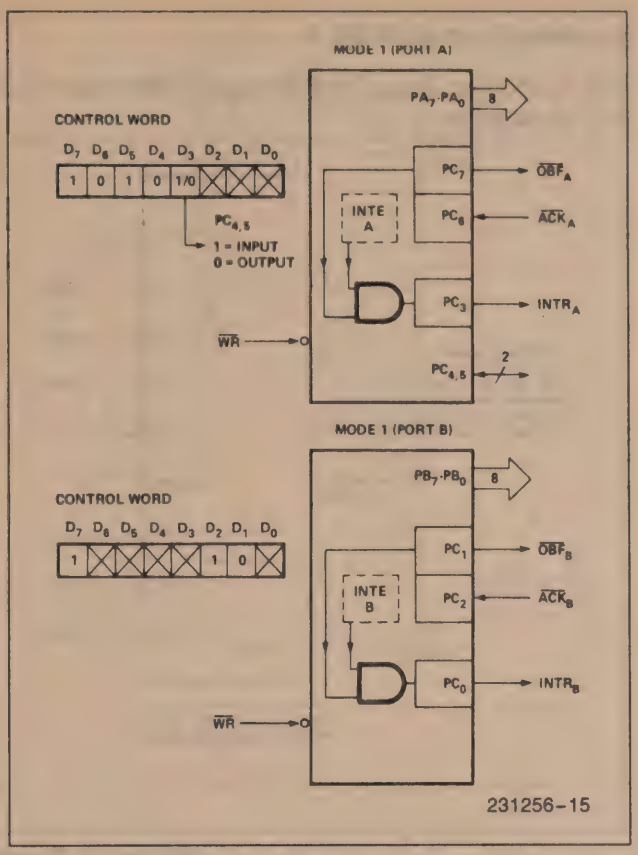


Figure 10. MODE 1 Output

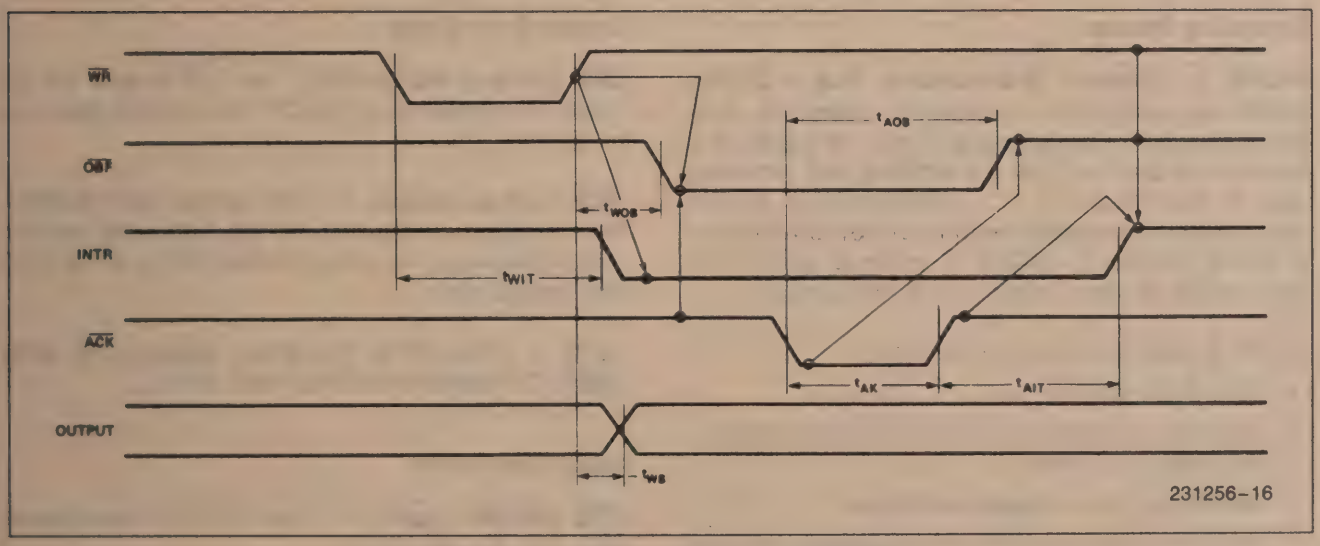


Figure 11. MODE 1 (Strobed Output)

Combinations of MODE 1

Port A and Port B can be individually defined as input or output in Mode 1 to support a wide variety of strobed I/O applications.

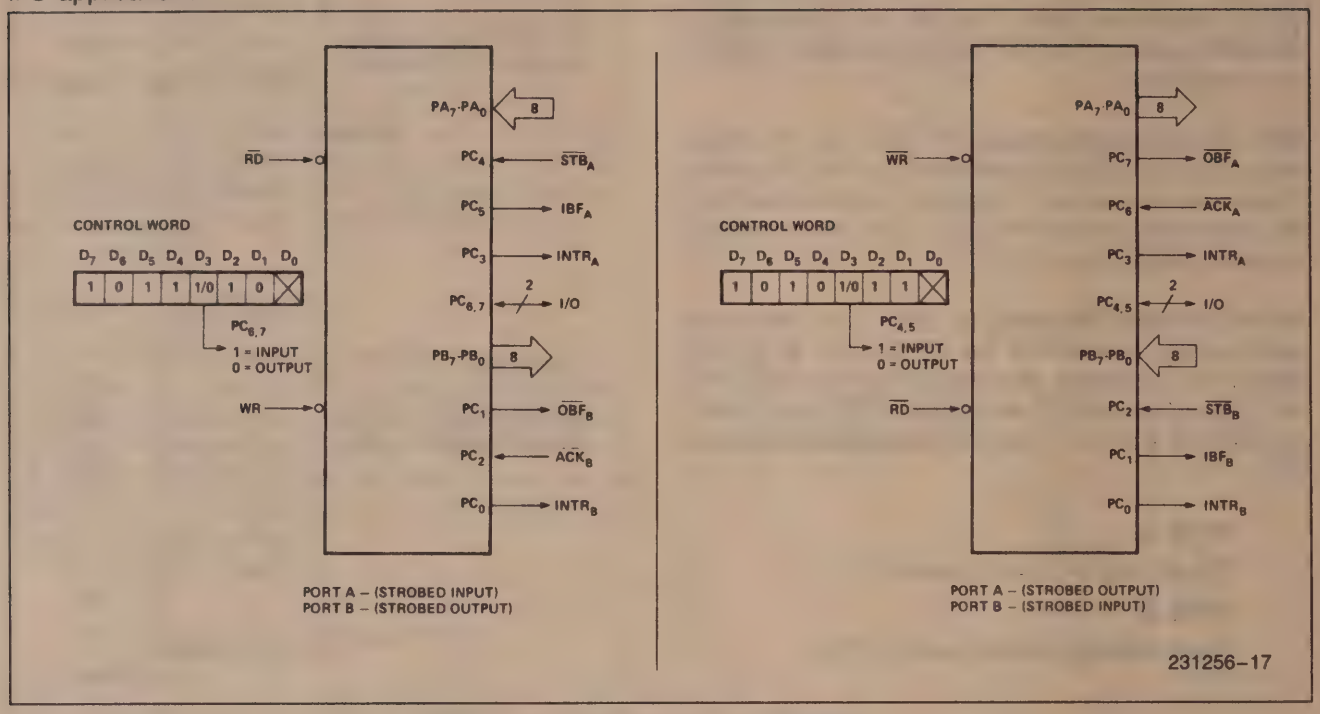


Figure 12. Combinations of MODE 1

Operating Modes

MODE 2 (Strobed Bidirectional Bus I/O). This functional configuration provides a means for communicating with a peripheral device or structure on a single 8-bit bus for both transmitting and receiving data (bidirectional bus I/O). "Handshaking" signals are provided to maintain proper bus flow discipline in a similar manner to MODE 1. Interrupt generation and enable/disable functions are also available.

MODE 2 Basic Functional Definitions:

- Used in Group A **only**.
- One 8-bit, bi-directional bus port (Port A) and a 5-bit control port (Port C).
- Both inputs and outputs are latched.
- The 5-bit control port (Port C) is used for control and status for the 8-bit, bi-directional bus port (Port A).

Bidirectional Bus I/O Control Signal Definition

INTR (Interrupt Request). A high on this output can be used to interrupt the CPU for input or output operations.

Output Operations

$\overline{OBF}$ (Output Buffer Full). The $\overline{OBF}$ output will go "low" to indicate that the CPU has written data out to port A.

$\overline{ACK}$ (Acknowledge). A "low" on this input enables the tri-state output buffer of Port A to send out the data. Otherwise, the output buffer will be in the high impedance state.

INTE 1 (The INTE Flip-Flop Associated with $\overline{OBF}$). Controlled by bit set/reset of PC_6.

Input Operations

$\overline{STB}$ (Strobe Input). A "low" on this input loads data into the input latch.

IBF (Input Buffer Full F/F). A "high" on this output indicates that data has been loaded into the input latch.

INTE 2 (The INTE Flip-Flop Associated with IBF). Controlled by bit set/reset of PC_4.

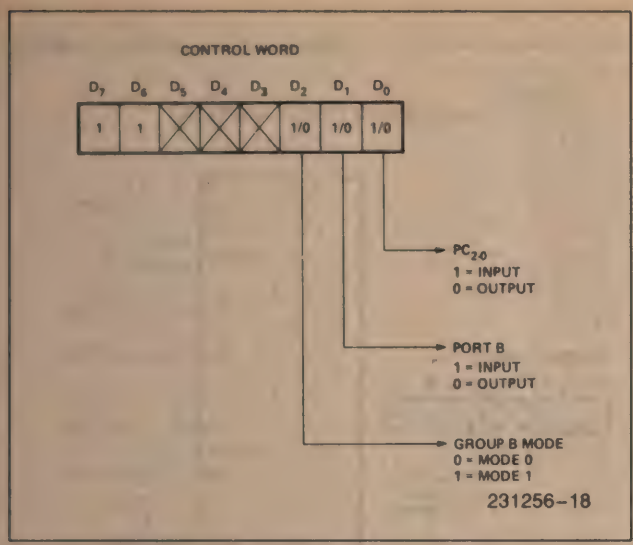


Figure 13. MODE Control Word

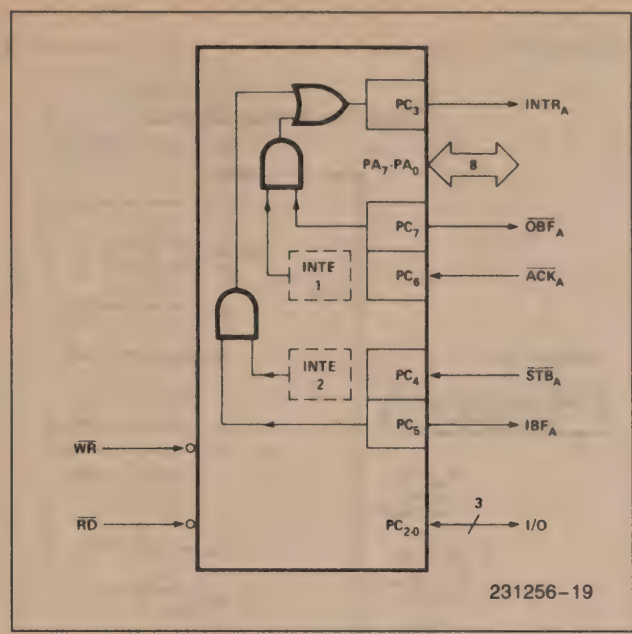


Figure 14. MODE 2

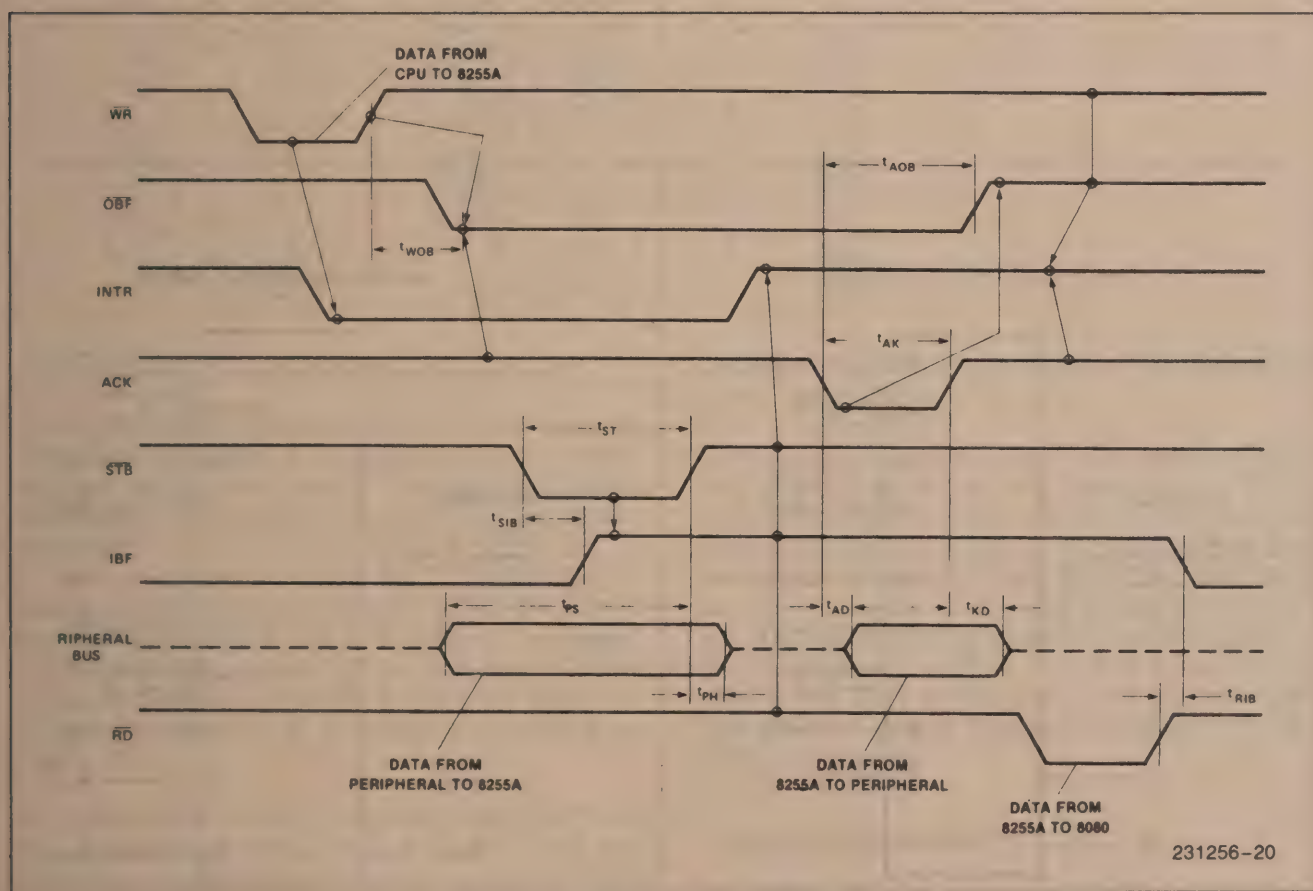


Figure 15. MODE 2 (Bidirectional)

NOTE:
Any sequence where $\overline{WR}$ occurs before $\overline{ACK}$, and $\overline{STB}$ occurs before $\overline{RD}$ is permissible.
(INTR = IBF • $\overline{MASK}$ • $\overline{STB}$ • $\overline{RD}$ + $\overline{OBF}$ • $\overline{MASK}$ • $\overline{ACK}$ • $\overline{WR}$)

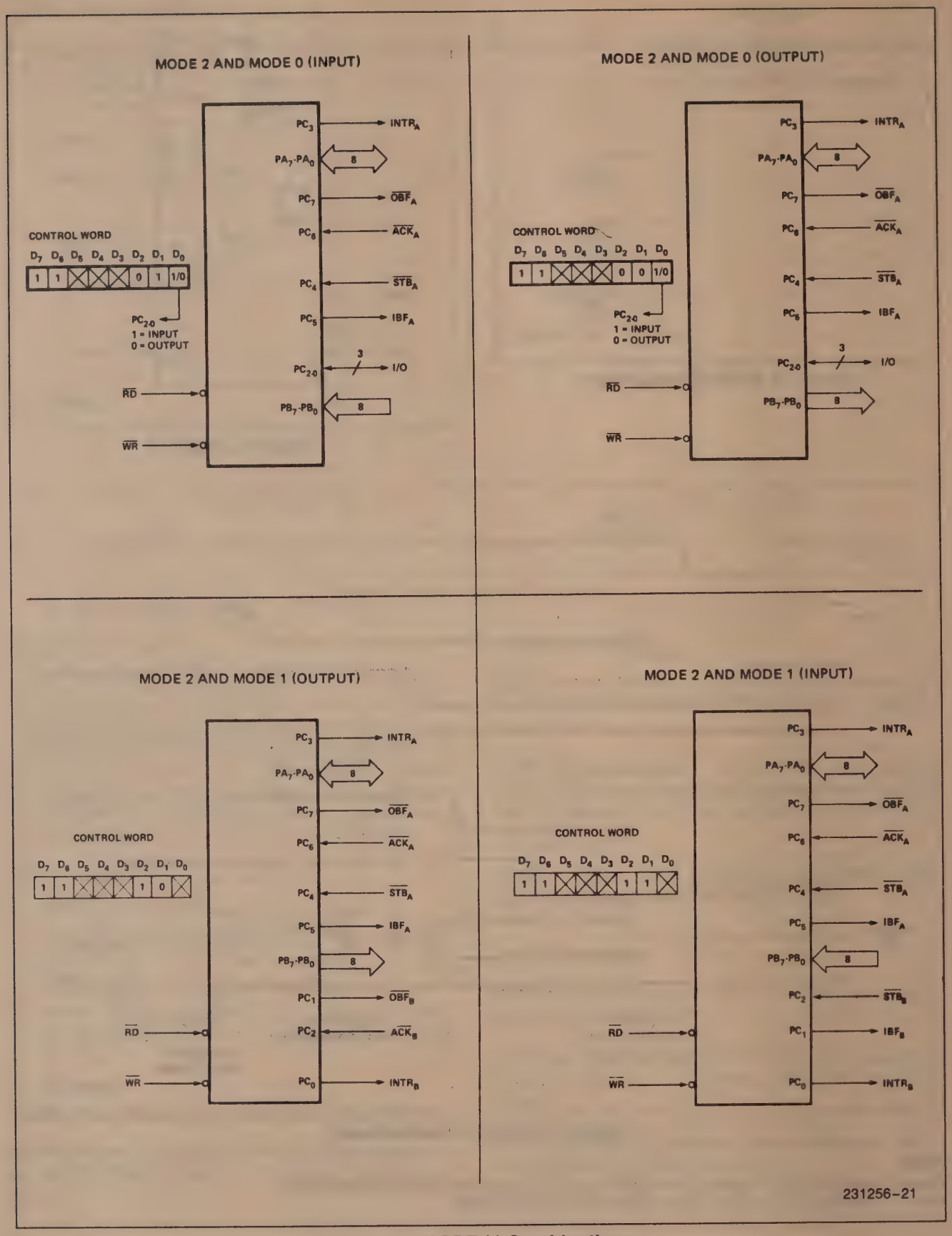


Figure 16. MODE ¼ Combinations

Mode Definition Summary

	MODE 0		MODE 1		MODE 2	
	IN	OUT	IN	OUT	GROUP A ONLY	
PA_0	IN	OUT	IN	OUT	↔	
PA_1	IN	OUT	IN	OUT	↔	
PA_2	IN	OUT	IN	OUT	↔	
PA_3	IN	OUT	IN	OUT	↔	
PA_4	IN	OUT	IN	OUT	↔	
PA_5	IN	OUT	IN	OUT	↔	
PA_6	IN	OUT	IN	OUT	↔	
PA_7	IN	OUT	IN	OUT	↔	
PB_0	IN	OUT	IN	OUT	—	MODE 0 OR MODE 1 ONLY
PB_1	IN	OUT	IN	OUT	—	
PB_2	IN	OUT	IN	OUT	—	
PB_3	IN	OUT	IN	OUT	—	
PB_4	IN	OUT	IN	OUT	—	
PB_5	IN	OUT	IN	OUT	—	
PB_6	IN	OUT	IN	OUT	—	
PB_7	IN	OUT	IN	OUT	—	
PC_0	IN	OUT	$INTR_B$	$INTR_B$	I/O	
PC_1	IN	OUT	IBF_B	$\overline{OBF}_B$	I/O	
PC_2	IN	OUT	$\overline{STB}_B$	$\overline{ACK}_B$	I/O	
PC_3	IN	OUT	$INTR_A$	$INTR_A$	$INTR_A$	
PC_4	IN	OUT	$\overline{STB}_A$	I/O	$\overline{STB}_A$	
PC_5	IN	OUT	IBF_A	I/O	IBF_A	
PC_6	IN	OUT	I/O	$\overline{ACK}_A$	$\overline{ACK}_A$	
PC_7	IN	OUT	I/O	$\overline{OBF}_A$	$\overline{OBF}_A$	

Special Mode Combination Considerations

There are several combinations of modes possible. For any combination, some or all of the Port C lines are used for control or status. The remaining bits are either inputs or outputs as defined by a "Set Mode" command.

During a read of Port C, the state of all the Port C lines, except the $\overline{ACK}$ and $\overline{STB}$ lines, will be placed on the data bus. In place of the $\overline{ACK}$ and $\overline{STB}$ line states, flag status will appear on the data bus in the PC2, PC4, and PC6 bit positions as illustrated by Figure 18.

Through a "Write Port C" command, only the Port C pins programmed as outputs in a Mode 0 group can be written. No other pins can be affected by a "Write Port C" command, nor can the interrupt enable flags be accessed. To write to any Port C output programmed as an output in a Mode 1 group or to change an interrupt enable flag, the "Set/Reset Port C Bit" command must be used.

With a "Set/Reset Port C Bit" command, any Port C line programmed as an output (including INTR, IBF and $\overline{OBF}$) can be written, or an interrupt enable flag can be either set or reset. Port C lines programmed as inputs, including $\overline{ACK}$ and $\overline{STB}$ lines, associated with Port C are not affected by a "Set/Reset Port C Bit" command. Writing to the corresponding Port C bit positions of the $\overline{ACK}$ and $\overline{STB}$ lines with the "Set/Reset Port C Bit" command will affect the Group A and Group B interrupt enable flags, as illustrated in Figure 18.

Current Drive Capability

Any output on Port A, B or C can sink or source 2.5 mA. This feature allows the 82C55A to directly drive Darlington type drivers and high-voltage displays that require such sink or source current.

Reading Port C Status

In Mode 0, Port C transfers data to or from the peripheral device. When the 82C55A is programmed to function in Modes 1 or 2, Port C generates or accepts "hand-shaking" signals with the peripheral device. Reading the contents of Port C allows the programmer to test or verify the "status" of each peripheral device and change the program flow accordingly.

There is no special instruction to read the status information from Port C. A normal read operation of Port C is executed to perform this function.

INPUT CONFIGURATION

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
I/O	I/O	IBF_A	$INTE_A$	$INTR_A$	$INTE_B$	IBF_B	$INTR_B$
GROUP A					GROUP B		

OUTPUT CONFIGURATIONS

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
$\overline{OBF}_A$	$INTE_A$	I/O	I/O	$INTR_A$	$INTE_B$	$\overline{OBF}_B$	$INTR_B$
GROUP A					GROUP B		

Figure 17a. MODE 1 Status Word Format

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
$\overline{OBF}_A$	$INTE_1$	IBF_A	$INTE_2$	$INTR_A$			
GROUP A					GROUP B		

(Defined By Mode 0 or Mode 1 Selection)

Figure 17b. MODE 2 Status Word Format

Interrupt Enable Flag	Position	Alternate Port C Pin Signal (Mode)
INTE B	PC2	$\overline{ACK}_B$ (Output Mode 1) or $\overline{STB}_B$ (Input Mode 1)
INTE A2	PC4	$\overline{STB}_A$ (Input Mode 1 or Mode 2)
INTE A1	PC6	$\overline{ACK}_A$ (Output Mode 1 or Mode 2

Figure 18. Interrupt Enable Flags in Modes 1 and 2

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias....0°C to + 70°C
Storage Temperature .....− 65°C to + 150°C
Supply Voltage .....− 0.5 to + 8.0V
Operating Voltage .....+ 4V to + 7V
Voltage on any Input.....GND−2V to + 6.5V
Voltage on any Output ..GND−0.5V to V_{CC} + 0.5V
Power Dissipation .....1 Watt

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

NOTICE: Specifications contained within the following tables are subject to change.

D.C. CHARACTERISTICS

T_A = 0°C to 70°C, V_{CC} = +5V ±10%, GND = 0V

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{IL}	Input Low Voltage	−0.5	0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC}	V	
V_{OL}	Output Low Voltage		0.4	V	I_{OL} = 2.5 mA
V_{OH}	Output High Voltage	3.0 V_{CC} − 0.4		V V	I_{OH} = −2.5 mA I_{OH} = −100 μA
I_{IL}	Input Leakage Current		±1	μA	V_{IN} = V_{CC} to 0V (Note 1)
I_{OFL}	Output Float Leakage Current		±10	μA	V_{IN} = V_{CC} to 0V (Note 2)
I_{DAR}	Darlington Drive Current	±2.5		mA	Ports A, B, C R_{ext} = 750Ω V_{ext} = 1.5V
I_{PHL}	Port Hold Low Leakage Current	+50	+300	μA	V_{OUT} = 1.0V Port A only
I_{PHH}	Port Hold High Leakage Current	−50	−300	μA	V_{OUT} = 3.0V Ports A, B, C
I_{PHLO}	Port Hold Low Overdrive Current	−350		μA	V_{OUT} = 0.8V
I_{PHHO}	Port Hold High Overdrive Current	+350		μA	V_{OUT} = 3.0V
I_{CC}	V_{CC} Supply Current		10	mA	(Note 3)
I_{CCSB}	V_{CC} Supply Current-Standby		10	μA	V_{CC} = 5.5V V_{IN} = V_{CC} or GND Port Conditions If I/P = Open/High O/P = Open Only With Data Bus = High/Low $\overline{CS}$ = High Reset = Low Pure Inputs = Low/High

NOTES:
1. Pins A_1, A_0, $\overline{CS}$, $\overline{WR}$, $\overline{RD}$, Reset.
2. Data Bus; Ports B, C.
3. Outputs open.

CAPACITANCE

T_A = 25°C, V_{CC} = GND = 0V

Symbol	Parameter	Min	Max	Units	Test Conditions
C_{IN}	Input Capacitance		10	pF	Unmeasured pins returned to GND
$C_{I/O}$	I/O Capacitance		20	pF	

A.C. CHARACTERISTICS

T_A = 0° to 70°C, V_{CC} = +5V ±10%, GND = 0V

BUS PARAMETERS

READ CYCLE

Symbol	Parameter	82C55A		82C55A-2		Units	Test Conditions
		Min	Max	Min	Max		
t_{AR}	Address Stable Before $\overline{RD}$ ↓	0		0		ns	
t_{RA}	Address Hold Time After $\overline{RD}$ ↑	0		0		ns	
t_{RR}	$\overline{RD}$ Pulse Width	150		150		ns	
t_{RD}	Data Delay from $\overline{RD}$ ↓		120		120	ns	
t_{DF}	$\overline{RD}$ ↑ to Data Floating	10	75	10	75	ns	
t_{RV}	Recovery Time between $\overline{RD}$/$\overline{WR}$	200		200		ns	

WRITE CYCLE

Symbol	Parameter	82C55A		82C55A-2		Units	Test Conditions
		Min	Max	Min	Max		
t_{AW}	Address Stable Before $\overline{WR}$ ↓	0		0		ns	
t_{WA}	Address Hold Time After $\overline{WR}$ ↑	20		20		ns	Ports A & B
		20		20		ns	Port C
t_{WW}	$\overline{WR}$ Pulse Width	100		100		ns	
t_{DW}	Data Setup Time Before $\overline{WR}$ ↑	100		100		ns	
t_{WD}	Data Hold Time After $\overline{WR}$ ↑	30		30		ns	Ports A & B
		30		30		ns	Port C

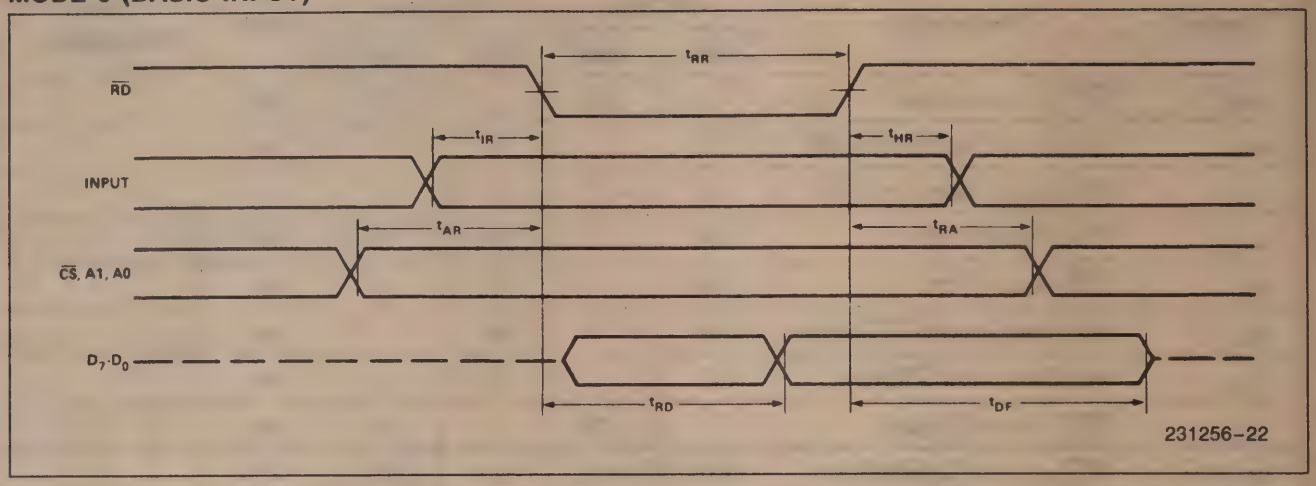
OTHER TIMINGS

Symbol	Parameter	82C55A		82C55A-2		Units	Test Conditions
		Min	Max	Min	Max		
t_{WB}	$\overline{WR}$ = 1 to Output		350		350	ns	
t_{IR}	Peripheral Data Before $\overline{RD}$	0		0		ns	
t_{HR}	Peripheral Data After $\overline{RD}$	0		0		ns	
t_{AK}	$\overline{ACK}$ Pulse Width	200		200		ns	
t_{ST}	$\overline{STB}$ Pulse Width	100		100		ns	
t_{PS}	Per. Data Before $\overline{STB}$ High	20		20		ns	
t_{PH}	Per. Data After $\overline{STB}$High	50		50		ns	
t_{AD}	$\overline{ACK}$ = 0 to Output		175		175	ns	
t_{KD}	$\overline{ACK}$ = 1 to Output Float	20	250	20	250	ns	
t_{WOB}	$\overline{WR}$ = 1 to $\overline{OBF}$ = O		150		150	ns	
t_{AOB}	$\overline{ACK}$ = 0 to $\overline{OBF}$ = 1		150		150	ns	
t_{SIB}	$\overline{STB}$ = 0 to IBF = 1		150		150	ns	
t_{RIB}	$\overline{RD}$ = 1 to IBF = 0		150		150	ns	
t_{RIT}	$\overline{RD}$ = 0 to INTR = 0		200		200	ns	
t_{SIT}	$\overline{STB}$ = 1 to INTR = 1		150		150	ns	
t_{AIT}	$\overline{ACK}$ = 1 to INTR = 1		150		150	ns	
t_{WIT}	$\overline{WR}$ = 0 to INTR = 0		200		200	ns	see note 1
t_{RES}	Reset Pulse Width	500		500		ns	see note 2

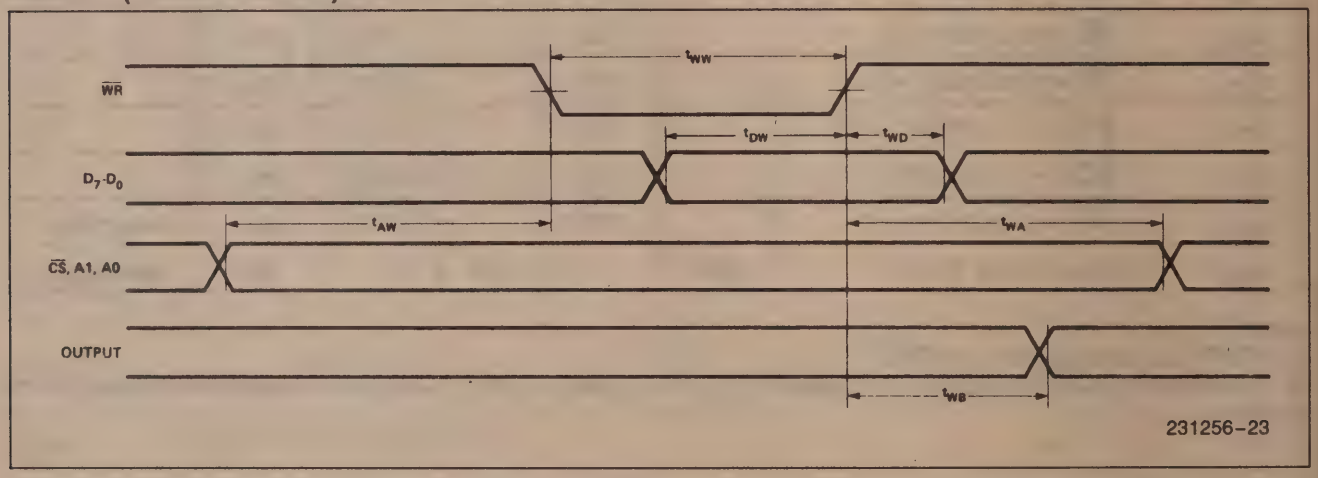
NOTE:
1. INTR ↑ may occur as early as $\overline{WR}$ ↓.
2. Pulse width of initial Reset pulse after power on must be at least 50 μSec. Subsequent Reset pulses may be 500 ns minimum.

WAVEFORMS

MODE 0 (BASIC INPUT)

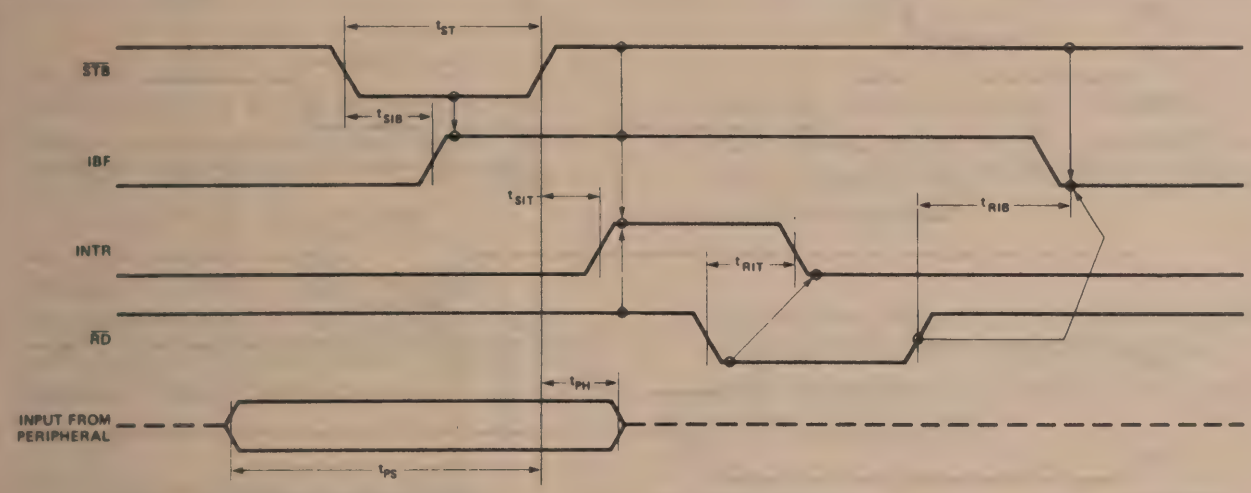


MODE 0 (BASIC OUTPUT)



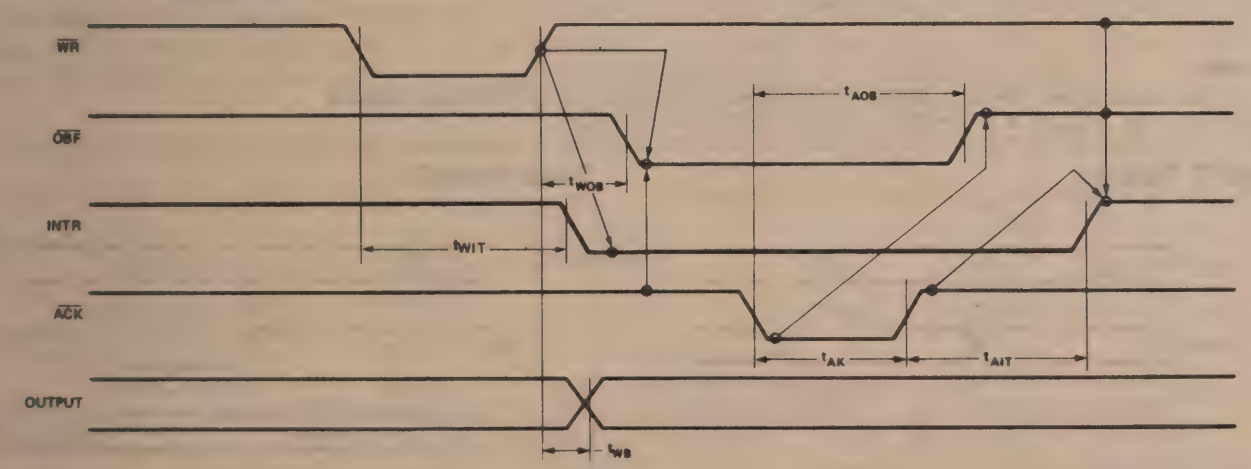
WAVEFORMS (Continued)

MODE 1 (STROBED INPUT)



231256-24

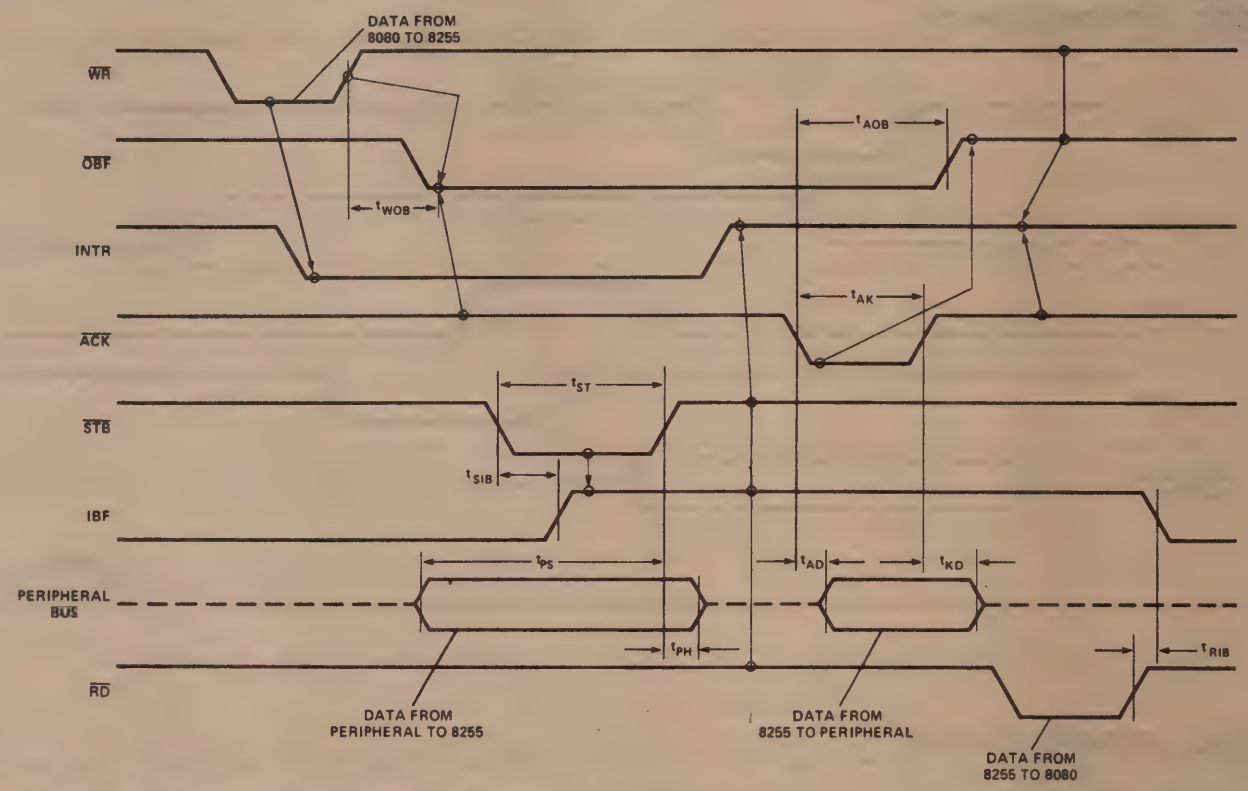
MODE 1 (STROBED OUTPUT)



231256-25

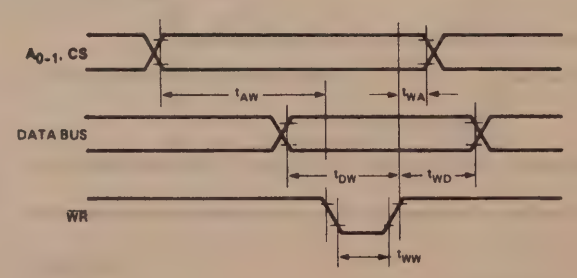
WAVEFORMS (Continued)

MODE 2 (BIDIRECTIONAL)

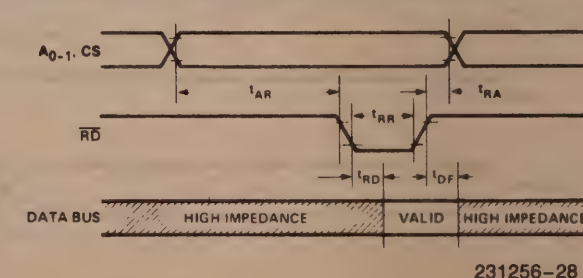


Note:
Any sequence where $\overline{WR}$ occurs before $\overline{ACK}$ AND $\overline{STB}$ occurs before $\overline{RD}$ is permissible.
(INTR = IBF • $\overline{MASK}$ • $\overline{STB}$ • $\overline{RD}$ + $\overline{OBF}$ • $\overline{MASK}$ • $\overline{ACK}$ • $\overline{WR}$)

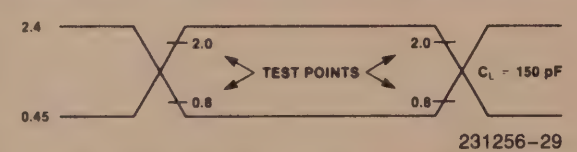
WRITE TIMING



READ TIMING

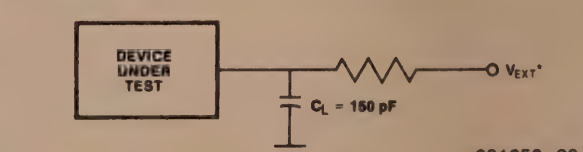


A.C. TESTING INPUT, OUTPUT WAVEFORM



A.C. Testing Inputs Are Driven At 2.4V For A Logic 1 And 0.45V For A Logic 0 Timing Measurements Are Made At 2.0V For A Logic 1 And 0.8 For A Logic 0.

A.C. TESTING LOAD CIRCUIT



*V_{EXT} Is Set At Various Voltages During Testing To Guarantee The Specification. C_L Includes Jig Capacitance.

PRELIMINARY

8256AH
MULTIFUNCTION MICROPROCESSOR SUPPORT CONTROLLER

- **Programmable Serial Asynchronous Communications Interface for 5-, 6-, 7-, or 8-Bit Characters, 1, 1½, or 2 Stop Bits, and Parity Generation**
- **On-Board Baud Rate Generator Programmable for 13 Common Baud Rates up to 19.2K Bits/second, or an External Baud Clock Maximum of 1M Bit/second**
- **Five 8-Bit Programmable Timer/Counters; Four Can Be Cascaded to Two 16-Bit Timer/Counters**
- **Two 8-Bit Programmable Parallel I/O Ports; Port 1 Can Be Programmed for Port 2 Handshake Controls and Event Counter Inputs**
- **Eight-Level Priority Interrupt Controller Programmable for 8085 or iAPX 86, iAPX 88 Systems and for Fully Nested Interrupt Capability**
- **Programmable System Clock to 1 ×, 2 ×, 3 ×, or 5 × 1.024 MHz**

The Intel® 8256AH Multifunction Universal Asynchronous Receiver-Transmitter (MUART) combines five commonly used functions into a single 40-pin device. It is designed to interface to the 8086/88, iAPX 186/188, and 8051 to perform serial communications, parallel I/O, timing, event counting, and priority interrupt functions. All of these functions are fully programmable through nine internal registers. In addition, the five timer/counters and two parallel I/O ports can be accessed directly by the microprocessor.

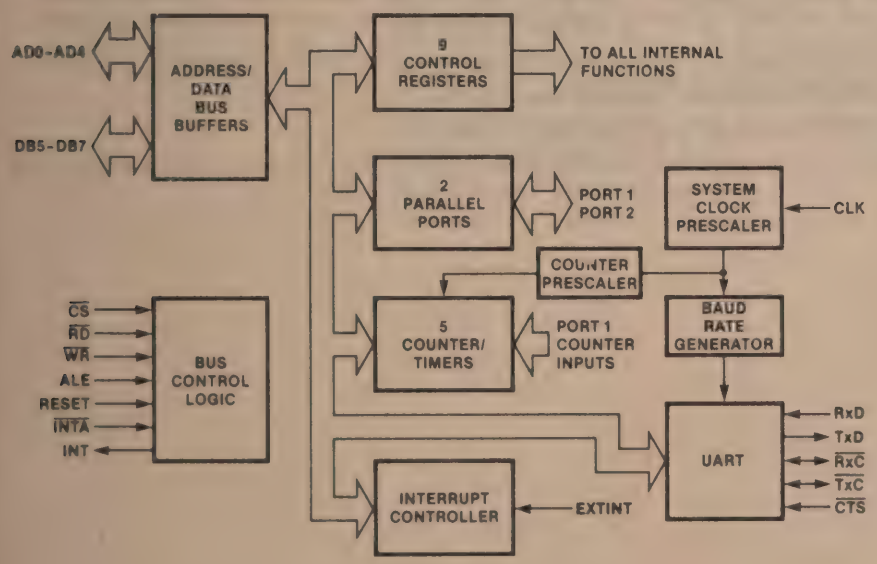


Figure 1. MUART Block Diagram

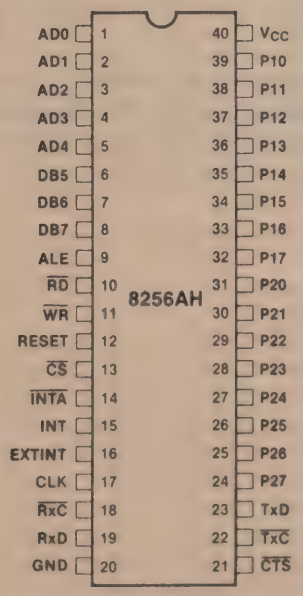


Figure 2. MUART Pin Configuration

Intel Corporation Assumes No Responsibility for the Use of Any Circuitry Other Than Circuitry Embodied in an Intel Product No Other Circuit Patent Licenses are Implied
© INTEL CORPORATION 1984

September 1984
ORDER NUMBER: 230759-002

Table 1. Pin Description

Symbol	Pin	Type	Name and Function
AD0-AD4 DB5-DB7	1-5 6-8	I/O	**ADDRESS/DATA:** Three-state address/data lines which interface to the lower 8 bits of the microprocessor's multiplexed address/data bus. The 5-bit address is latched on the falling edge of ALE. In the 8-bit mode, AD0-AD3 are used to select the proper register, while AD1-AD4 are used in the 16-bit mode. AD4 in the 8-bit mode is ignored as an address, while AD0 in the 16-bit mode is used as a second chip select, active low.
ALE	9	I	**ADDRESS LATCH ENABLE:** Latches the 5 address lines on AD0-AD4 and $\overline{CS}$ on the falling edge.
$\overline{RD}$	10	I	**READ CONTROL:** When this signal is low, the selected register is gated onto the data bus.
$\overline{WR}$	11	I	**WRITE CONTROL:** When this signal is low, the value on the data bus is written into the selected register.
RESET	12	I	**RESET:** An active high pulse on this pin forces the chip into its initial state. The chip remains in this state until control information is written.
$\overline{CS}$	13	I	**CHIP SELECT:** A low on this signal enables the MUART. It is latched with the address on the falling edge of ALE, and $\overline{RD}$ and $\overline{WR}$ have no effect unless $\overline{CS}$ was latched low during the ALE cycle.
$\overline{INTA}$	14	I	**INTERRUPT ACKNOWLEDGE:** If the MUART has been enabled to respond to interrupts, this signal informs the MUART that its interrupt request is being acknowledged by the microprocessor. During this acknowledgement the MUART puts an RSTn instruction on the data bus for the 8-bit mode or a vector for the 16-bit mode.
INT	15	O	**INTERRUPT REQUEST:** A high signals the microprocessor that the MUART needs service.
EXTINT	16	I	**EXTERNAL INTERRUPT:** An external device can request interrupt service through this input. The input is level sensitive (high), therefore it must be held high until an $\overline{INTA}$ occurs or the interrupt address register is read.
CLK	17	I	**SYSTEM CLOCK:** The reference clock for the baud rate generator and the timers.
RxC	18	I/O	**RECEIVE CLOCK:** If the baud rate bits in the Command Register 2 are all 0, this pin is an input which clocks serial data into the RxD pin on the rising edge of RxC. If baud rate bits in Command Register 2 are programmed from 1-0FH, this pin outputs a square wave whose rising edge indicates when the data on RxD is being sampled. This output remains high during start, stop, and parity bits.
RxD	19	I	**RECEIVE DATA:** Serial data input.
GND	20	PS	**GROUND:** Power supply and logic ground reference.

Table 1. Pin Description (continued)

Symbol	Pin	Type	Name and Function
$\overline{CTS}$	21	I	**CLEAR TO SEND:** This input enables the serial transmitter. If 1, 1.5, or 2 stop bits are selected $\overline{CTS}$ is level sensitive. As long as $\overline{CTS}$ is low, any character loaded into the transmitter buffer register will be transmitter serially. A single negative going pulse causes the transmission of a single character previously loaded into the transmitter buffer register. If a baud rate from 1-OFH is selected, $\overline{CTS}$ must be low for at least 1/32 of a bit, or it will be ignored. If the transmitter buffer is empty, this pulse will be ignored. If this pulse occurs during the transmission of a character up to the time where ½ the first (or only) stop bit is sent out, it will be ignored. If it occurs afterwards, but before the end of the stop bits, the next character will be transmitted immediately following the current one. If $\overline{CTS}$ is still high when the transmitter register is sending the last stop bit, the transmitter will enter its idle state until the next high-to-low transition on $\overline{CTS}$ occurs. If 0.75 stop bits is chosen, the $\overline{CTS}$ input is edge sensitive. A negative edge on $\overline{CTS}$ results in the immediate transmission of the next character. The length of the stop bits is determined by the time interval between the beginning of the first stop bit and the next negative edge on $\overline{CTS}$. A high-to-low transition has no effect if the transmitter buffer is empty or if the time interval between the beginning of the stop bit and next negative edge is less than 0.75 bits. A high or a low level or a low-to-high transition has no effect on the transmitter for the 0.75 stop bit mode.
TxC	22	I/O	**TRANSMIT CLOCK:** If the baud rate bits in command register 2 are all set to 0, this input clocks data out of the transmitter on the falling edge. If baud rate bits are programmed for 1 or 2, this input permits the user to provide a 32x or 64x clock which is used for the receiver and transmitter. If the baud rate bits are programmed for 3-0FH, the internal transmitter clock is output. As an output it delivers the transmitter clock at the selected bit rate. If 1½ or 0.75 stop bits are selected, the transmitter divider will be asynchronously reset at the beginning of each start bit, immediately causing a high-to-low transition on TxC. TxC makes a high-to-low transition at the beginning of each serial bit, and a low-to-high transition at the center of each bit.
TxD	23	O	**TRANSMIT DATA:** Serial data output.
P27-P20	24-31	I/O	**PARALLEL I/O PORT 2:** Eight bit general purpose I/O port. Each nibble (4 bits) of this port can be either an input or an output. The outputs are latched whereas the input signals are not. Also, this port can be used as an 8-bit input or output port when using the two-wire handshake. In the handshake mode both inputs and outputs are latched.
P17-P10	32-39	I/O	**PARALLEL I/O PORT 1:** Each pin can be programmed as an input or an output to perform general purpose I/O. All outputs are latched whereas inputs are not. Alternatively these pins can serve as control pins which extend the functional spectrum of the chip.
V_{CC}	40	PS	**POWER:** +5V power supply.

FUNCTIONAL DESCRIPTION

The 8256AH Multi-Function Universal Asynchronous Receiver-Transmitter (MUART) combines five commonly used functions into a single 40-pin device. The MUART performs asynchronous serial communications, parallel I/O, timing, event counting, and interrupt control. For detailed application information, see Intel Ap Note #153, Designing with the 8256.

Serial Communications

The serial communications portion of the MUART contains a full-duplex asynchronous receiver-transmitter (UART). A programmable baud rate generator is included on the MUART to permit a variety of operating speeds without external components. The UART can be programmed by the CPU for a variety of character sizes, parity generation and detection, error detection, and start/stop bit handling. The receiver checks the start and stop bits in the center of the bit, and a break halts the reception of data. The transmitter can send breaks and can be controlled by an external enable pin.

Parallel I/O

The MUART includes 16 bits of general purpose parallel I/O. Eight bits (Port 1) can be individually changed from input to output or used for special I/O functions. The other eight bits (Port 2) can be used as nibbles (4 bits) or as bytes. These eight bits also include a handshaking capability using two pins on Port 1.

Counter/Timers

There are five 8-bit counter/timers on the MUART. The timers can be programmed to use either a 1 kHz or 16 kHz clock generated from the system clock. Four of the 8-bit counter/timers can be cascaded to two 16-bit counter/timers, and one of the 8-bit counter/timers can be reset to its initial value by an external signal.

Interrupts

An eight-level priority interrupt controller can be configured for fully nested or normal interrupt priority. Seven of the eight interrupts service functions on the MUART (counter/timers, UART), and one external interrupt is provided which can be used for a particular function or for chaining interrupt controllers or more MUARTs. The MUART will support 8085 and 8086/88 systems with direct interrupt vectoring, or the MUART can be polled to determine the cause of the interrupt. If additional interrupt control capability is needed, the MUART's interrupt controller can be cascaded into another MUART, into an Intel 8259A Programmable Interrupt Controller, or into the interrupt controller of the iAPX 186/188 High-Integration Microprocessor.

INITIALIZATION

In general the MUART's functions are independent of each other and only the registers and bits associated with a particular function need to be initialized, not the entire chip. The command sequence is arbitrary since every register is directly addressable; however, Command Byte 1 must be loaded first. To put the device into a fully operational condition, it is necessary to write the following commands:

Command byte 1
Command byte 2
Command byte 3
Mode byte
Port 1 control
Set Interrupts

The modification register may be loaded if required for special applications; normally this operation is not necessary. The MUART should be reset before initialization. (Either a hardware or a software reset will do.)

INTERFACING

This section describes the hardware interface between the 8256 MUART and the 80186 microprocessor. Figure 3 displays the block diagram for this interface. The MUART can be interfaced to many other microprocessors using these basic principles.

In all cases the 8256 will be connected directly to the CPU's multiplexed address/data bus. If latches or data bus buffers are used in a system, the MUART should be on the microprocessor side of the address/data bus. The MUART latches the address internally on the falling edge of ALE. The address consists of Chip Select (CS) and four address lines. For 8-bit microprocessors, AD0-AD3 are the address lines. For 16-bit microprocessors, AD1-AD4 are the address lines; AD0 is used as a second chip select which is active low. Since chip select is internally latched along with the address, it does not have to remain active during the entire instruction cycle. As long as the chip select setup and hold times are met, it can be derived from multiplexed address/data lines or multiplexed address/status lines. When the 8256 is in the 16-bit mode, A0 serves as a second chip select. As a result the MUART's internal registers will all have even addresses since A0 must be zero to select the device. Normally the MUART will be placed on the lower data byte. If the MUART is placed on the upper data byte.

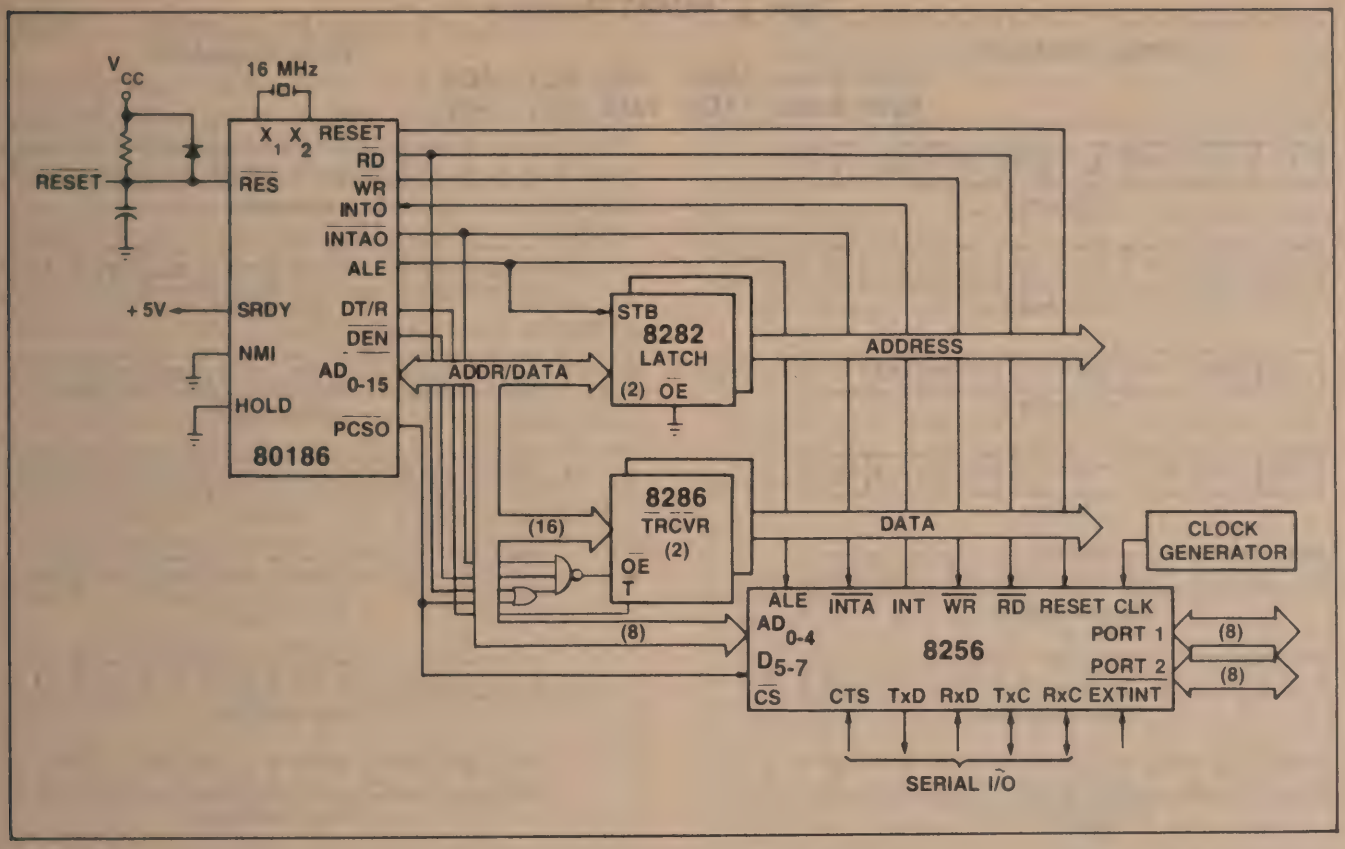


Figure 3. 80186/8256 Interface

the internal registers will be 512 address locations apart and the chip would occupy an 8 K word address space.

DESCRIPTION OF THE REGISTERS

The following section will provide a description of the registers and define the bits within the registers where appropriate. Table 2 lists the registers and their addresses.

Command Register 1

L1	L0	S1	S0	BRKI	BITI	8086	FRQ
(0R)					(0W)		

FRQ — Timer Frequency Select

This bit selects between two frequencies for the five timers. If FRQ = 0, the timer input frequency is 16 kHz (62.5μs). If FRQ = 1, the timer input frequency is 1 KHz (1 ms). The selected clock frequency is shared by all the counter/timers enabled for timing; thus, all timers must run with the same time base.

8086 — 8086 Mode Enable

This bit selects between 8085 mode and 8086/8088 mode. In 8085 mode (8086 = 0), A0 to A3 are used to address the internal registers, and an RSTn instruction is generated in response to the first INTA. In In 8086 mode (8086 = 1), A1 to A4 are used to address the internal registers, and A0 is used as an extra chip select (A0 must equal zero to be enabled). The response to INTA is for 8086 interrupts where the first INTA is ignored, and an interrupt vector (40H to 47H is placed on the bus in response to the second INTA.

BITI — Interrupt on Bit Change

This bit selects between one of two interrupt sources on Priority Level 1, either Counter/Timer 2 or Port 1 P17 interrupt. When this bit equals 0, Counter/Timer 2 will be mapped into Priority Level 1. If BITI equals 0 and Level 1 interrupt is enabled, a transition from 1 to 0 in Counter/Timer 2 will generate an interrupt request on Level 1. When BITI equals 1, Port 1 P17 external edge triggered interrupt source is mapped into Priority Level 1. In this case if Level 1 is enabled, a low-to-high transition on P17 generates an interrupt request on Level 1.

Table 2. MUART Registers

Read Registers								8085 Mode: AD3	AD2	AD1	AD0	Write Registers							
								8086 Mode: AD4	AD3	AD2	AD1								
L1	L0	S1	S0	BRKI	BITI	8086	FRQ	0	0	0	0	L1	L0	S1	S0	BRKI	BITI	8086	FRQ
Command 1												Command 1							
PEN	EP	C1	C0	B3	B2	B1	B0	0	0	0	1	PEN	EP	C1	C0	B3	B2	B1	B0
Command 2												Command 2							
0	RxE	IAE	NIE	0	SBRK	TBRK	0	0	0	1	0	SET	RxE	IAE	NIE	END	SBRK	TBRK	RST
Command 3												Command 3							
T35	T24	T5C	CT3	CT2	P2C2	P2C1	P2C0	0	0	1	1	T35	T24	T5C	CT3	CT2	P2C2	P2C1	P2C0
Mode												Mode							
P17	P16	P15	P14	P13	P12	P11	P10	0	1	0	0	P17	P16	P15	P14	P13	P12	P11	P10
Port 1 Control												Port 1 Control							
L7	L6	L5	L4	L3	L2	L1	L0	0	1	0	1	L7	L6	L5	L4	L3	L2	L1	L0
Interrupt Enable												Set Interrupts							
D7	D6	D5	D4	D3	D2	D1	D0	0	1	1	0	L7	L6	L5	L4	L3	L2	L1	L0
Interrupt Address												Reset Interrupts							
D7	D6	D5	D4	D3	D2	D1	D0	0	1	1	1	D7	D6	D5	D4	D3	D2	D1	D0
Receiver Buffer												Transmitter Buffer							
D7	D6	D5	D4	D3	D2	D1	D0	1	0	0	0	D7	D6	D5	D4	D3	D2	D1	D0
Port 1												Port 1							
D7	D6	D5	D4	D3	D2	D1	D0	1	0	0	1	D7	D6	D5	D4	D3	D2	D1	D0
Port 2												Port 2							
D7	D6	D5	D4	D3	D2	D1	D0	1	0	1	0	D7	D6	D5	D4	D3	D2	D1	D0
Timer 1												Timer 1							
D7	D6	D5	D4	D3	D2	D1	D0	1	0	1	1	D7	D6	D5	D4	D3	D2	D1	D0
Timer 2												Timer 2							
D7	D6	D5	D4	D3	D2	D1	D0	1	1	0	0	D7	D6	D5	D4	D3	D2	D1	D0
Timer 3												Timer 3							
D7	D6	D5	D4	D3	D2	D1	D0	1	1	0	1	D7	D6	D5	D4	D3	D2	D1	D0
Timer 4												Timer 4							
D7	D6	D5	D4	D3	D2	D1	D0	1	1	1	0	D7	D6	D5	D4	D3	D2	D1	D0
Timer 5												Timer 5							
INT	RBF	TBE	TRE	BD	PE	OE	FE	1	1	1	1	0	RS4	RS3	RS2	RS1	RS0	TME	DSC
Status												Modification							

BRKI — Break-In Detect Enable

If this bit equals 0, Port 1 P16 is a general purpose I/O port. When BRKI equals 1, the Break-In Detect feature is enabled on Port 1 P16. A Break-In condition is present on the transmission line when it is forced to the start bit voltage level by the receiving station. Port 1 P16 must be connected externally to the transmission line in order to detect a Break-In. A Break-In is polled by the MUART during the transmission of the last or only stop bit of a character.

A Break-In Detect is OR-ed with Break Detect in Bit 3 of the Status Register. The distinction can be made through the interrupt controller. If the transmit and receive interrupts are enabled, a Break-In will generate an interrupt on Level 5, the transmit interrupt, while Break will generate an interrupt on Level 4, the receive interrupt.

S0, S1 — Stop Bit Length

S1	S0	Stop Bit Length
0	0	1
0	1	1.5
1	0	2
1	1	0.75

The relationship of the number of stop bits and the function of input $\overline{CTS}$ is discussed in the Pin Description section under "$\overline{CTS}$".

L0, L1 — Character Length

L1	L0	Character Length
0	0	8
0	1	7
1	0	6
1	1	5

Command Register 2

PEN	EP	C1	C0	B3	B2	B1	B0
(1R)				(1W)			

Programming bits 0. . .3 with values from 3H to FH enables the internal baud rate generator as a common clock source for the transmitter and receiver and determines its divider ratio.

Programming bits 0. . .3 with values of 1H or 2H enables input TxC as a common clock source for the transmitter and receiver. The external clock must provide a frequency of either 32x or 64x the baud rate. The data transmission rates range from 0. . .32 Kbaud.

If bits 0. . .3 are set to 0, separate clocks must be input to pin RxC for the receiver and pin TxC for the transmitter. Thus, different baud rates can be used for transmission and reception. In this case, prescalers are disabled and the input serial clock frequency must match the baud rate. The input serial clock frequency can range from 0 to 1.024 MHz.

B0, B1, B2, B3 — Baud Rate Select

These four bits select the bit clock's source, sampling rate, and serial rate for the internal baud rate generator.

B3	B2	B1	B0	Baud Rate	Sampling Rate
0	0	0	0	$\overline{TxC}$, $\overline{RxC}$	1
0	0	0	1	$\overline{TxC}$/64	64
0	0	1	0	$\overline{TxC}$/32	32
0	0	1	1	19200	32
0	1	0	0	9600	64
0	1	0	1	4800	64
0	1	1	0	2400	64
0	1	1	1	1200	64
1	0	0	0	600	64
1	0	0	1	300	64
1	0	1	0	200	64
1	0	1	1	150	64
1	1	0	0	110	64
1	1	0	1	100	64
1	1	1	0	75	64
1	1	1	1	50	64

The following table gives an overview of the function of pins TxC and RxC:

Bits 3 to 0 (Hex.)	TxC	RxC
0	Input: 1 x baud rate clock for the transmitter	Input: 1 x baud rate clock for the receiver
1, 2	Input: 32 x or 64 x baud rate for transmitter and receiver	Output: receiver bit clock with a low-to-high transition at data bit sampling time. Otherwise: high level
3 to F	Output: baud rate clock of the transmitter	Output: as above

As an output, RxC outputs a low-to-high transition at sampling time of every data bit of a character. Thus, data can be loaded, e.g., into a shift register externally. The transition occurs only if data bits of a character are present. It does not occur for start, parity, and stop bits (RxC = high).

As an output, TxC outputs the internal baud rate clock of the transmitter. There will be a high-to-low transition at every beginning of a bit.

C0, C1 — System Clock Prescaler (Bits 4, 5)

Bits 4 and 5 define the system clock prescaler divider ratio. The internal operating frequency of 1.024 MHz is derived from the system clock.

C1	C0	Divider Ratio	Clock at Pin CLK
0	0	5	5.12 MHz
0	1	3	3.072 MHz
1	0	2	2.048 MHz
1	1	1	1.024 MHz

EP — Even Parity (Bit 6)

EP = 0: Odd parity
EP = 1: Even parity

PEN — Parity Enable (Bit 7)

Bit 7 enables parity generation and checking.

PEN = 0: No parity bit
PEN = 1: Enable parity bit

The parity bit according to Command Register 2 bit 6 (see above) is inserted between the last data bit of a character and the first or only stop bit. The parity bit is checked during reception. A false parity bit generates an error indication in the Status Register and an Interrupt Request on Level 4.

Command Register 3

SET	RxE	IAE	NIW	END	SBRK	TBRK	RST
	(2R)				(2W)		

Command Register 3 is different from the first two registers because it has a bit set/reset capability. Writing a byte with Bit 7 high sets any bits which were also high. Writing a byte with Bit 7 low resets any bits which were high. If any bit 0-6 is low, no change occurs to that bit. When Command Register 3 is read, bits 0, 3, and 7 will always be zero.

RST — Reset

If RST is set, the following events occur:

1. All bits in the Status Register except bits 4 and 5 are cleared, and bits 4 and 5 are set.
2. The Interrupt Enable, Interrupt Request, and Interrupt Service Registers are cleared. Pending requests and indications for interrupts in service will be cancelled. Interrupt signal INT will go low.
3. The receiver and transmitter are reset. The transmitter goes idle (TxD is high), and the receiver enters start bit search mode.
4. If Port 2 is programmed for handshake mode, IBF and OBF are reset high.

RST does *not* alter ports, data registers or command registers, but it halts any operation in progress. RST is automatically cleared.

RST = 0 has not effect. The reset operation triggered by Command Register 3 is a subset of the hardware reset.

TBRK — Transmit Break

The transmission data output TxD will be set low as soon as the transmission of the previous character has been finished. It stays low until TBRK is cleared. The state of $\overline{CTS}$ is of no significance for this operation. As long as break is active, data transfer from the Transmitter Buffer to the Transmitter Register will be inhibited. As soon as TBRK is reset, the break condition will be deactivated and the transmitter will be re-enabled.

SBRK — Single Character Break

This causes the transmitter data to be set low for one character including start bit, data bits, parity bit, and stop bits. SBRK is automatically cleared when time for the last data bit has passed. It will start after the character in progress completes, and will delay the next data transfer from the Transmitter Buffer to the Transmitter Register until TxD returns to an idle (marking) state. If both TBRK and SBRK are set, break will be set as long as TBRK is set, but SBRK will be cleared after one character time of break. If SBRK is set again, it remains set for another character. The user can send a definite number of break characters in this manner by clearing TBRK after setting SBRK for the last character time.

END — End of Interrupt

If fully nested interrupt mode is selected, this bit reset the currently served interrupt level in the Interrupt Service Register. *This command must occur at the end of each interrupt service routine during fully nested interrupt mode.* END is automatically cleared when the Interrupt Service Register (internal) is cleared. END is ignored if nested interrupts are not enabled.

NIE — Nested Interrupt Enable

When NIE equals 1, the interrupt controller will operate in the nested interrupt mode. When NIE equals 0, the interrupt controller will operate in the normal interrupt mode. Refer to the "Interrupt controller" section of AP-153 under "Normal Mode" and "Nested Mode" for a detailed description of these operations.

IAE — Interrupt Acknowledge Enable

This bit enables an automatic response to $\overline{\text{INTA}}$. The particular response is determined by the 8086 bit in Command Register 1.

RxE — Receive Enable

This bit enables the serial receiver and its associated status bits in the status register. If this bit is reset, the serial receiver will be disabled and the receive status bits will not be updated.

Note that the detection of break characters remains enabled while the receiver is disabled; i.e., Status Register Bit 3 (BD) will be set while the receiver is disabled whenever a break character has been recognized at the receive data input RxD.

SET — Bit Set/Reset

If this bit is high during a write to Command Register 3, then any bit marked by a high will set. If this bit is low, then any bit marked by a high will be cleared.

Mode Register

T35	T24	T5C	CT3	CT2	P2C2	P2C1	P2C0
(3R)					(3W)		

If test mode is selected, the output from the internal baud rate generator is placed on bit 4 of Port 1 (pin 35).

To achieve this, it is necessary to program bit 4 of Port 1 as an output (Port 1 Control Register Bit P14 = 1), and to program Command Register 2 bits B3 - B0 with a value ⩾ 3H.

P2C2, P2C1, P2C0 — Port 2 Control

P2C2	P2C1	P2C0	Mode	Direction Upper	Direction Lower
0	0	0	Nibble	Input	Input
0	0	1	Nibble	Input	Output
0	1	0	Nibble	Output	Input
0	1	1	Nibble	Output	Output
1	0	0	Byte Handshake	Input	
1	0	1	Byte Handshake	Output	
1	1	0	*DO NOT USE*		
1	1	1	Test		

NOTE:
If Port 2 is operating in handshake mode, Interrupt Level 7 is not available for Timer 5. Instead it is assigned to Port 2 handshaking.

CT2, CT3 — Counter/Timer Mode

Bit 3 and 4 defines the mode of operation of event counter/timers 2 and 3 regardless of its use as a single unit or as a cascaded one.

If CT2 or CT3 are high, then counter/timer 2 or 3 respectively is configured as an event counter on bit 2 or 3 respectively of Port 1 (pins 37 or 36). The event counter decrements the count by one on each low-to-high transition of the external input. If CT2 or CT3 is low, then the respective counter/timer is configured as a timer and the Port 1 pins are used for parallel I/O.

T5C — Timer 5 Control

If T5C is set, then Timer 5 can be preset and started by an external signal. Writing to the Timer 5 register loads the Timer 5 save register and stops the timer. A high-to-low transition on bit 5 of Port 1 (pin 34) loads the timer with the saved value and starts the timer. The next high-to-low transition on pin 34 retriggers the timer by reloading it with the initial value and continues timing.

Following a hardware reset, the save register is reset to 00H and both clock and trigger inputs are disabled. Transferring an instruction with T5C = 1 enables the trigger input; the save register can now be loaded with an initial value. The first trigger pulse causes the initial value to be loaded from the save register and enables the counter to count down to zero.

When the timer reaches zero it issues an interrupt request, disables its interrupt level and continues counting. A subsequent high-to-low transition on pin 5 resets Timer 5 to its initial value. For another timer interrupt, the Timer 5 interrupt enable bit must be set again.

T35, T24 — Cascade Timers

These two bits cascade Timers 3 and 5 or 2 and 4. Timers 2 and 3 are the lower bytes, while Timers 4 and 5 are the upper bytes. If T5C is set, then both Timers 3 and 5 can be preset and started by an external pulse.

When a high-to-low transition occurs, Timer 5 is preset to its saved value, But Timer 3 is always preset to all ones. If either CT2 or CT3 is set, then the corresponding timer pair is a 16-bit event counter.

A summary of the counter/timer control bits is given in Table 3.

NOTE:
Interrupt levels assigned to single counters are partly not occupied if event counters/timers are cascaded. Level 2 will be vacated if event counters/timers 2 and 4 are cascaded. Likewise, Level 7 will be vacated if event counters/timers 3 and 5 are cascaded.

Single event counters/timers generate an interrupt request on the transition from 01H to 00H, while cascaded ones generate it on the transition from 0001H to 0000H.

Port 1 Control Register

P17	P16	P15	P14	P13	P12	P11	P10
(4W)					(4W)		

Each bit in the Port 1 Control Register configures the direction of the corresponding pin. If the bit is high, the pin is an output, and if it low the pin is an input. Every Port 1 pin has another function which is controlled by other registers. If that special function is disabled, the pin functions as a general I/O pin as specified by this register. The special functions for each pin are described below.

Port 10, 11 — Handshake Control

If byte handshake control is enabled for Port 2 by the Mode Register, then Port 10 is programmed as $\overline{STB}/\overline{ACK}$ handshake control input, and Port 11 is programmed as $\overline{IBF}/\overline{OBF}$ handshake control output.

If byte handshake mode is enabled for output on Port 2 $\overline{OBF}$ indicates that a character has been loaded

Table 3. Event Counters/Timers Mode of Operation

Event Counter/ Timer	Function	Programming (Mode Word)	Clock Source
1	8-bit timer	—	Internal clock
2	8-bit timer	T24=0, CT2=0	Internal clock
	8-bit event counter	T24=0, CT2=1	P12 pin 37
2	8-bit timer	T35=0, CT3=0	Internal clock
	8-bit event counter	T35=0, CT3=1	P13 pin 36
4	8-bit timer	T24=0	Internal clock
5	8-bit timer, normal mode	T35=0, T5C=0	Internal clock
	8-bit timer, retriggerable mode	T35=0, T5C=1	Internal clock
2 and 4 cascaded	16-bit timer	T24=1, CT2=0	Internal clock
	16-bit event counter	T24=1, CT2=1	P12 pin 37
3 and 5 cascaded	16-bit timer, normal mode	T35=1, T5C=0, CT3=0	Internal clock
	16-bit event counter, normal mode	T35=1, T5C=0, CT3=1	P13 pin 36
	16-bit timer, retriggerable mode	T35=1, T5C=1, CT3=0	Internal clock
	16-bit event counter, retriggerable mode	T35=1, T5C=1, CT3=1	P13 pin 36

into the Port 2 output buffer. When an external device reads the data, it acknowledges this operation by driving $\overline{ACK}$ low. $\overline{OBF}$ is set low by writing to Port 2 and is reset by $\overline{ACK}$.

If byte handshake mode is enabled for input on Port 2, $\overline{STB}$ is an input. $\overline{IBF}$ is driven low after $\overline{STB}$ goes low. On the rising edge of $\overline{STB}$ the data from Port 2 is latched.

$\overline{IBF}$ is reset high when Port 2 is read.

Port 12, 13 — Counter 2, 3 Input

If Timer 2 or Timer 3 is programmed as an event counter by the Mode Register, then Port 12 or Port 13 is the counter input for Event Counter 2 or 3, respectively.

Port 14 — Baud Rate Generator Output Clock

If test mode is enabled by the Mode Register and Command Register 2 baud rate select is greater than 2, then Port 14 is an output from the internal baud rate generator.

P14 in Port 1 control register must be set to 1 for the baud rate generator clock to be output. The baud rate generator clock is 64 x the serial bit rate except at 19.2Kbps when it is 32 x the bit rate.

Port 15 — Timer 5 Trigger

If T5C is set in the Mode Register enabling a retriggerable timer, then Port 15 is the input which starts and reloads Timer 5.

A high-to-low transition on P15 (Pin 34) loads the timer with the save register and starts the timer.

Port 16 — Break-In Detect

If Break-In Detect is enabled by BRKI in Command Register 1, then this input is used to sense a Break-In. If Port 16 is low while the serial transmitter is sending the last stop bit, then a Break-In condition is signaled.

Port 17 — Port Interrupt Source

If BITI in Command Register 1 is set, then a low-to-high transition on Port 17 generates an interrupt request on Priority Level 1.

Port 17 is edge triggered.

Interrupt Enable Register

L7	L6	L5	L4	L3	L2	L1	L0

(5R) (5W=enable, (6W=disable)

Interrupts are enabled by writing to the Set Interrupts Register (5W). Interrupts are disabled by writing to the Reset Interrupts Register (6W). Each bit set by the Set Interrupts Register (5W) will enable that level interrupt, and each bit set in the Reset Interrupts Register (6W) will disable that level interrupt. The user can determine which interrupts are enabled by reading the Interrupt enable Register (5R).

Priority		Source
Highest	L0	Timer 1
	L1	Timer 2 or Port Interrupt
	L2	External Interrupt (EXTINT)
	L3	Timer 3 or Timers 3 & 5
	L4	Receiver Interrupt
	L5	Transmitter Interrupt
	L6	Timer 4 or Timers 2 & 4
Lowest	L7	Timer 5 or Port 2 Handshaking

Interrupt Address Register

0	0	0	D4	D3	D2	0	0

Interrupt Level Indication

(6R)

Reading the interrupt address register transfers an identifier for the currently requested interrupt level on the system data bus. This identifier is the number of the interrupt level multiplied by 4. It can be used by the CPU as an offset address for interrupt handling. Reading the interrupt address register has the same effect as a hardware interrupt acknowledge INTA; it clears the interrupt request pin (INT) and indicates an interrupt acknowledgement to the interrupt controller.

Receiver and Transmitter Buffer

D7	D6	D5	D4	D3	D2	D1	D0

(7R) (7W)

Both the receiver and transmitter in the MUART are double buffered. This means that the transmitter and receiver have a shift register and a buffer register. The buffer registers are directly addressable by reading or writing to register seven. After the receiver buffer is full, the RBF bit in the status register is set.

Reading the receive buffer clears the RBF status bit. The transmit buffer should be written to only if the TBE bit in the status register is set. Bytes written to the transmit buffer are held there until the transmit shift register is empty, assuming $\overline{CTS}$ is low. If the transmit buffer and shift register are empty, writing to the transmit buffer immediately transfers the byte to the transmit shift register. If a serial character length is less than 8 bits, the unused most significant bits are set to zero when reading the receive buffer, and are ignored when writing to the transmit buffer.

Port 1

D7	D6	D5	D4	D3	D2	D1	D0
(8R)					(8W)		

Writing to Port 1 sets the data in the Port 1 output latch. Writing to an input pin does not affect the pin, but the data is stored and will be output if the direction of the pin is changed later. If the pin is used as a control signal, the pin will not be affected, but the data is stored. Reading Port 1 transfers the data in Port 1 onto the data bus.

Port 2

D7	D6	D5	D4	D3	D2	D1	D0
(9R)					(9W)		

Writing to Port 2 sets the data in the Port 2 output latch. Writing to an input pin does not affect the pin, but it does store the data in the latch. Reading Port 2 puts the input pins onto the bus or the contents of the output latch for output pins.

Timer 1-5

D7	D6	D5	D4	D3	D2	D1	D0
($0A_{16}$-$0E_{16}$R)					($0A_{16}$-$0E_{16}$W)		

Reading Timer N puts the contents of the timer onto the data bus. If the counter changes while $\overline{RD}$ is low, the value on the data bus will not change. If two timers are cascaded, reading the high-order byte will cause the low-order byte to be latched. Reading the low-order byte will unlatch them both. Writing to either timer or decascading them also clears the latch condition. Writing to a timer sets the starting value of that timer. If two timers are cascaded, writing to the high-order byte presets the low-order byte to all ones. Loading only the high-order byte with a value of X leads to a count of X *256 + 255. Timers count down continuously. If the interrupt is enabled, it occurs when the counter changes from 1 to 0.

The timer/counter interrupts are automatically disabled when the interrupt request is generated.

Status Register

INT	RBF	TBE	TRE	BD	PE	OE	FE
				($0F_{16}$R)			

Reading the status register gates its contents onto the data bus. It holds the operational status of the serial interface as well as the status of the interrupt pin INT. The status register can be read at any time. The flags are stable and well defined at all instants.

FE — Framing Error, Transmission Mode

Bit 0 can be used in two modes. Normally, FE indicates framing error which can be changed to transmission mode indication by setting the TME bit in the modification register.

If transmission mode is disabled (in Modification Register), then FE indicates a framing error. A framing error is detected during the *first* stop bit. The error is reset by reading the Status Register or by a chip reset. A framing error does not inhibit the loading of the Receiver Buffer. If RxD remains low, the receiver will assemble the next character. The false stop bit is treated as the next start bit, and no high-to-low transition on RxD is required to synchronize the receiver.

When the TME bit in the Modification Register is set, FE is used to indicate that the transmitter was active during the reception of a character, thus indicating that the character received was transmitted by its own transmitter. FE is reset when the transmitter is not active during the reception of character. Reading the status register will not reset the FE bit in the transmission mode.

OE — Overrun Error

If the user does not read the character in the Receiver Buffer before the next character is received and transferred to this register, then the OE bit is set. The OE flag is set during the reception of the first stop bit and is cleared when the Status Register is read or when a hardware or software reset occurs. The first character received in this case will be lost.

PE — Parity Error

This bit indicates that a parity error has occurred during the reception of a character. A parity error is present if value of the parity bit in the received character is different from the one expected according to command word 2 bits 6 EP. The parity bit is expected and checked only if it is enabled by command word 2 bit 7 PEN.

A parity error is set during the first stop bit and is reset by reading the Status Register or by a chip reset.

BD — Break/Break-In

The BD bit flags whether a break character has been received, or a Break-In condition exists on the transmission line. Command Register 1 Bit 3 (BRKI) enables the Break-In Detect function.

Whenever a break character has been received, Status Register Bit 3 will be set and in addition an interrupt request on Level 4 is generated. The receiver will be idled. It will be started again with the next high-to-low transition at pin RxD.

The break character received will not be loaded into the receiver buffer register.

If Break-In Detection is enabled and a Break-In condition occurs, Status Register Bit 3 will be set and in addition an interrupt request on Level 5 is generated.

The BD status bit will be reset on reading the status register or on a hardware or software reset. For more information on Break/Break-In, refer to the "Serial Asynchronous Communication" section of AP-153 under "Receive Break Detect" and "Break-In Detect."

TRE — Transmit Register Empty

When TRE is set the transmit register is empty and an interrupt request is generated on Level 5 if enabled. When TRE equals 0 the transmit register is in the process of sending data. TRE is set by a chip reset and when the last stop bit has left the transmitter. It is reset when a character is loaded into the Transmitter Register. If $\overline{CTS}$ is low, the Transmitter Register will be loaded during the transmission of the start bit. If $\overline{CTS}$ is high at the end of a character, TRE will remain high and no character will be loaded into the Transmitter Register until $\overline{CTS}$ goes low. If the transmitter was inactive before a character is loaded into the Transmitter Buffer, the Transmitter Register will be empty temporarily while the buffer is full. However, the data in the buffer will be transferred to the transmitter register immediately and TRE will be cleared while TBE is set.

TBE — Transmitter Buffer Empty

TBE indicates the Transmitter Buffer is empty and is ready to accept a character. TBE is set by a chip reset or the transfer of data to the Transmitter Register, and is cleared when a character is written to the transmitter buffer. When TBE is set, an interrupt request is generated on Level 5 if enabled.

RBF — Receiver Buffer Full

RBF is set when the Receiver Buffer has been loaded with a new character during the sampling of the first stop bit. RBF is cleared by reading the receiver buffer or by a chip reset.

INT — Interrupt Pending

The INT bit reflects the state of the INT Pin (Pin 15) and indicates an interrupt is pending. It is reset by $\overline{INTA}$ or by reading the Interrupt Address Register if only one interrupt is pending and by a chip reset.

FE, OE, PE, RBF, and Break Detect all generate a Level 4 interrupt when the receiver samples the first stop bit. TRE, TBE, and Break-In Detect generate a Level 5 interrupt. TRE generates an interrupt when TBE is set and the Transmitter Register finished transmitting. The Break-In Detect interrupt is issued at the same time as TBE or TRE.

Modification Register

0	RS4	RS3	RS2	RS1	RS0	TME	DSC

($0F_{16}$W)

DSC — Disable Start Bit Check

DSC disables the receiver's start bit check. In this state the receiver will not be reset if RxD is not low at the center of the start bit.

TME — Transmission Mode Enable

TME enables transmission mode and disables framing error detection. For information on transmission mode see the description of the framing error bit in the Status Register.

RS0, RS1, RS2, RS3, RS4 — Receiver Sample Time

The number in RSn alters when the receiver samples RxD. The receiver sample time can be modified only if the receiver is *not* clocked by RxC.

NOTE:
The modification register cannot be read. Reading from address 0FH, 8086: 1EH gates the contents of the status register onto the data bus.

A hardware reset (reset, Pin 12) resets all modification register bits to 0, i.e.:
- The start bit check is enabled.
- Status Register Bit 0 (FE) indicates framing error.
- The sampling time of the serial receiver is the bit center.

A software reset (Command Word 3, RST) does not affect the modification register.

Hardware Reset

A reset signal on pin RESET (HIGH level) forces the device 8256 into a well-defined initial state. This state is characterized as follows:

1. Command registers 1, 2 and 3, mode register, Port 1 control register, and modification register are reset. Thus, all bits of the parallel interface are set to be inputs and event counters/timers are configured as independent 8-bit timers.

2. Status register bits are reset with the exception of bits 4 and 5. Bits 4 and 5 are set indicating that both transmitter register and transmitter buffer register are empty.

3. The interrupt mask, interrupt request, and interrupt service register bits are reset and disable all requests. As a consequence, interrupt signal INT IS INACTIVE (LOW).

4. The transmit data output is set to the marking state (HIGH) and the receiver section is disabled until it is enabled by Command Register 3 Bit 6.

5. The start bit will be checked at sampling time. The receiver will return to start bit search mode if input RxD is not LOW at this time.

6. Status Register Bit 0 implies framing error.

7. The receiver samples input RxD at bit center.

Reset has no effect on the contents of receiver buffer register, transmitter buffer register, the intermediate latches of parallel ports, and event counters/timers, respectively.

RS4	RS3	RS2	RS1	RS0	Point of time between start of bit and end of bit measured in steps of 1/32 bit length
0	1	1	1	1	1 (Start of Bit)
0	1	1	1	0	2
0	1	1	0	1	3
0	1	1	0	0	4
0	1	0	1	1	5
0	1	0	1	0	6
0	1	0	0	1	7
0	1	0	0	0	8
0	0	1	1	1	9
0	0	1	1	0	10
0	0	1	0	1	11
0	0	1	0	0	12
0	0	0	1	1	13
0	0	0	1	0	14
0	0	0	0	1	15
0	0	0	0	0	16 (Bit center)
1	1	1	1	1	17
1	1	1	1	0	18
1	1	1	0	1	19
1	1	1	0	0	20
1	1	0	1	1	21
1	1	0	1	0	22
1	1	0	0	1	23
1	1	0	0	0	24
1	0	1	1	1	25
1	0	1	1	0	26
1	0	1	0	1	27
1	0	1	0	0	28
1	0	0	1	1	29
1	0	0	1	0	30
1	0	0	0	1	31
1	0	0	0	0	32 (End of Bit)

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias	0° C to 70° C
Storage Temperature	-65° C to +150° C
Voltage On Any Pin With Respect to ground	-0.5V to +7V
Power Dissication	1 Watt

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = +5.0V ± 10%)

Symbol	Parameter	Min.	Max.	Units	Test Conditions
V_{IL}	Input Low Voltage	−0.5	0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC} + 0.5	V	
V_{OL}	Output Low Voltage		0.45	V	I_{OL} = 2.5 mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = −400 μA
I_{IL}	Input Leakage		10 −10	μA μA	V_{IN} = V_{CC} V_{IN} = 0V
I_{LO}	Output Leakage		10 −10	μA μA	V_{OUT} = V_{CC} V_{OUT} = 0.45V
I_{CC}	V_{CC} Supply Current		160	mA	

CAPACITANCE (T_A = 25°C, V_{CC} = GND = 0V)

Symbol	Parameter	Min.	Max.	Units	Test Conditions
C_{IN}	Input Capacitance		10	pF	f_c = 1 MHz
$C_{I/O}$	I/O Capacitance		20	pF	Unmeasured pins returned to V_{SS}

A.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{cc} = +5.0V ± 10%, GND = 0V)

BUS PARAMETERS

Symbol	Parameter	8256AH Min.	8256AH Max.	Units
tLL	ALE Pulse Width	50		ns
tCSL	$\overline{CS}$ to ALE Setup Time	0		ns
tAL	Address to ALE Setup Time	20		ns
tLA	Address Hold Time After ALE	25		ns
tLC	ALE to $\overline{RD}/\overline{WR}$	20		ns
tCC	$\overline{RD}$, $\overline{WR}$, $\overline{INTA}$ Pulse Width	200		ns
tRD	Data Valid from $\overline{RD}$ (1)		120	ns
tDF	Data Float After $\overline{RD}$ (2)		50	ns
tDW	Data Valid to $\overline{WR}$	150		ns
tWD	Data Valid After $\overline{WR}$	50		ns
tCL	$\overline{RD}/\overline{WR}$ Control to Latch Enable	25		ns
tLDR	ALE to Data Valid		150	ns
tRST	Reset Pulse Width	300		ns
tRV	Recovery Time Between $\overline{RD}/\overline{WR}$	500		ns

TIMER/COUNTER PARAMETERS

tCPI	Counter Input Cycle Time (P12, P13)	2.2		μs
tCPWH	Counter Input Pulse Width High	1.1		μs
tCPWL	Counter Input Pulse Width Low	1.1		μs
tTPI	Counter Input↑ to INT↑ at Terminal Count		2.75	μs
tTIH	LOAD Pulse High Time Counter 5	1.1		μs
tTIL	LOAD Pulse Low Time Counter 5	1.1		μs
tPP	Counter 5 Load Before Next Clock Pulse on P13	1.1		μs
tCR	External Count Clock↑ to $\overline{RD}$↓ to Ensure Clock is Reflected in Count	2.2		μs
tRC	$\overline{RD}$↑ to External Count Clock↑ to Ensure Clock is not Reflected in Count	0		ns
tCW	External Count Clock↑ ro $\overline{WR}$↑ to Ensure Count Written is Not Decremented	2.2		μs
tWC	$\overline{WR}$↑ to External Count Clock to Ensure Count Written is Decremented	0		ns

INTERRUPT PARAMETERS

tDEX	EXTINT↑ to INT↑		200	ns
tDPI	Interrupt request on P17↑ to INT↑		2tCY +500	ns
tPI	Pulse Width of Interrupt Request on P17	tCY+ 100		ns
tHEA	$\overline{INTA}$↑ or $\overline{RD}$↑ to EXTINT↓	30		ns
tHIA	$\overline{INTA}$↑ or $\overline{RD}$↑ to INT↓		300	ns

A.C. CHARACTERISTICS (continued)

SERIAL INTERFACE AND CLOCK PARAMETERS

Symbol	Parameter	8256AH		Units
		Min.	Max.	
tCY	Clock Period	195	1000	ns
tCLKH	Clock High Pulse Width	65		ns
tCLKL	Clock Low Pulse Width	65		ns
tR	Clock Rise Time		20	ns
tF	Clock Fall Time		20	ns
tSCY	Serial Clock Period (4)	975		ns
tSPD	Serial Clock High (4)	350		ns
tSPW	Serial Clock Low (4)	350		ns
tSTD	Internal Status Update Delay From Center of Stop Bit (5)		300	ns
tDTX	$\overline{TxC}$ to TxD Data Valid		300	ns
tIRBF	INT Delay From Center of First Stop Bit		2tCY +500	ns
tITBE	INT Delay From Falling Edge of Transmit Clock at end of Start Bit		2tCY +500	ns
tCTS	Pulse Width for Single Character Transmission	(6)		

PARALLEL I/O PORT PARAMETERS

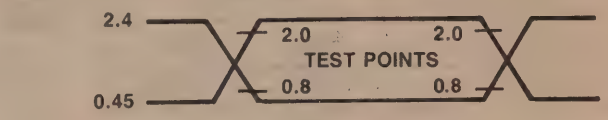
tWP	$\overline{WR}$ ↑ to P1/P2 Data Valid		0	ns
tPR	P1/P2 Data Stable Before $\overline{RD}$ ↓ (7)	300		ns
tRP	P1/P2 Data Hold Time	50		ns
tAK	$\overline{ACK}$ Pulse Width	150		ns
tST	Strobe Pulse Width	tSIB		ns
tPS	Data Setup to $\overline{STB}$ ↑	50		ns
tPH	Data Hold After $\overline{STB}$ ↑	50		ns
tWOB	$\overline{WR}$ ↑ to $\overline{OBF}$ ↑		250	ns
tAOB	$\overline{ACK}$↓ to $\overline{OBF}$↓		250	ns
tSIB	$\overline{STB}$ ↓ to $\overline{IBF}$ ↓		250	ns
tRI	$\overline{RD}$ ↑ to $\overline{IBF}$ ↑		250	ns
tSIT	$\overline{STB}$ ↑ to $\overline{INT}$ ↑		2tCY +500	ns
tAIT	$\overline{ACK}$ ↑ to $\overline{INT}$ ↑		2tCY +500	ns
tAED	$\overline{OBF}$↓ to $\overline{ACK}$ ↓ Delay	0		ns

NOTES:

1. C_L = pF all outputs.
2. Measured from logic "one" or "zero" to 1.5V at C_L = 150 pF.
3. P12, P13 are external clock inputs.
4. Note that RxC may be used as an input only in 1X mode, otherwise it will be an output.
5. The center of the Stop Bit will be the receiver sample time, as programmed by the modification register.
6. 1/16th bit length for 32X, 64X; 100 ns for 1X.
7. To ensure t_{RD} spec is met.

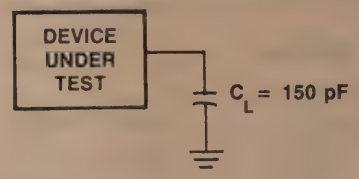
WAVEFORMS

A.C. TESTING INPUT, OUTPUT WAVEFORM



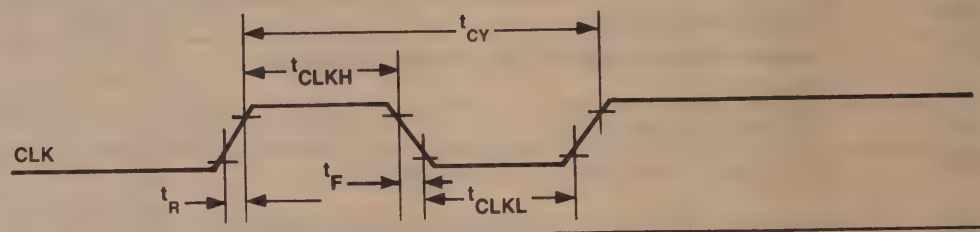
NOTES:
A.C. testing: inputs are driven at 2.4V for a logic "1" and 0.45V for a logic "0". timing measurements are made at 2.0V for a logic "1" and 0.8V for a logic "0".

A.C. TESTING LOAD CIRCUIT

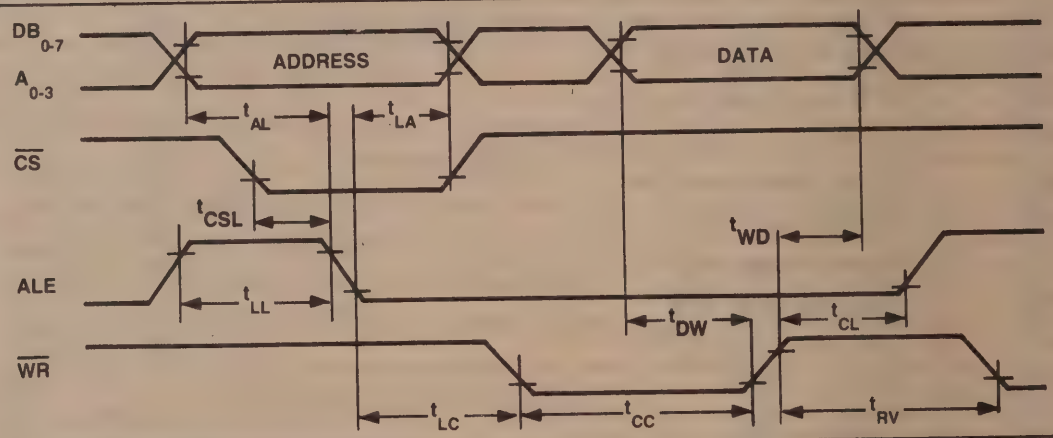


NOTES:
C_L = 150 pF
C_L includes jig capacitance

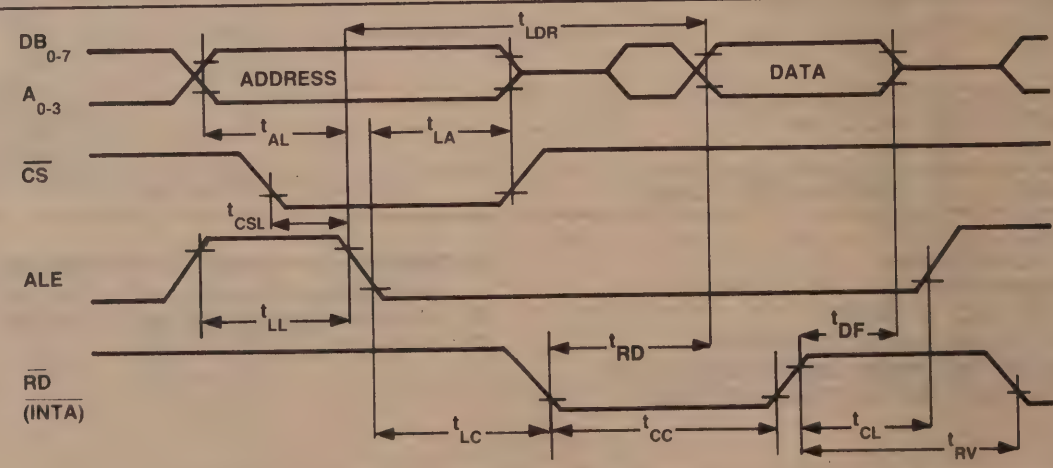
SYSTEM CLOCK



WRITE CYCLE



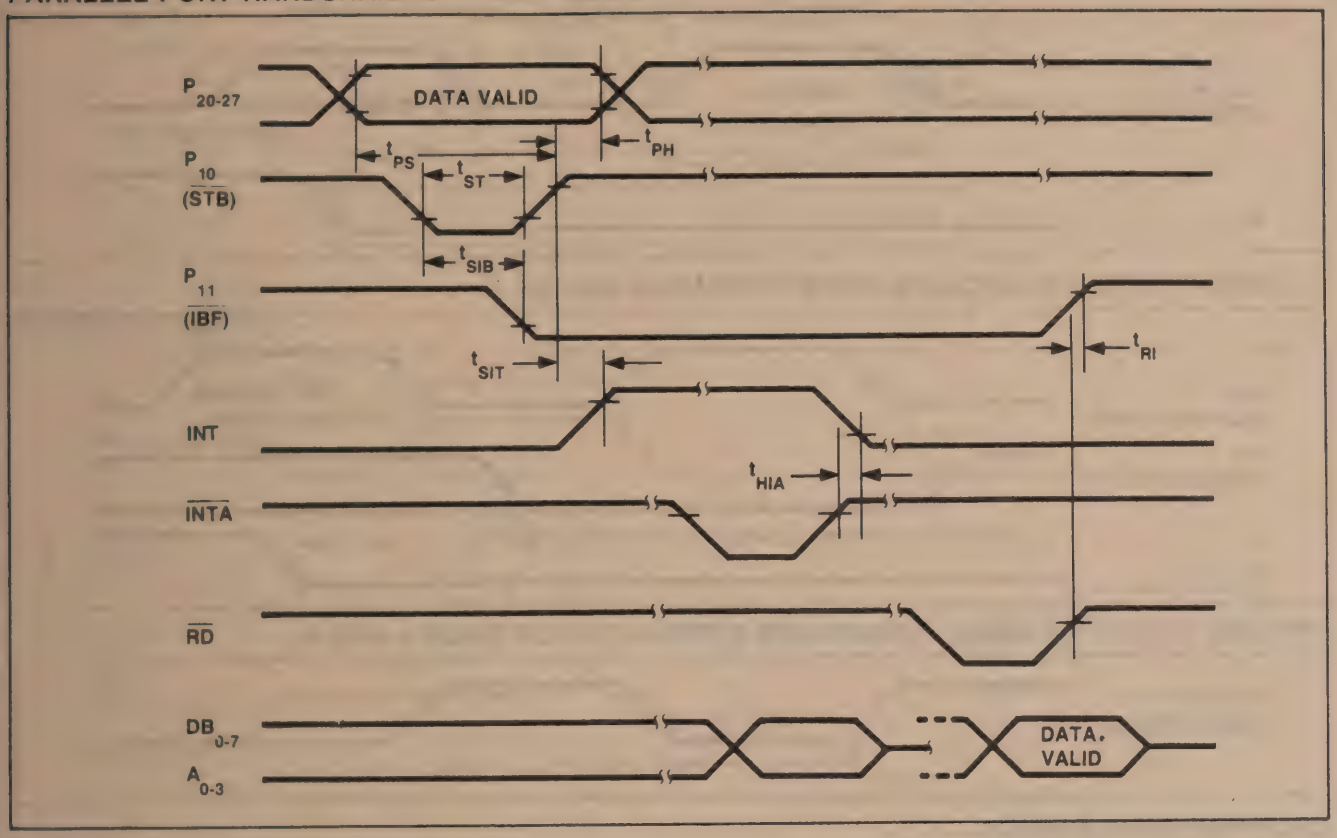
READ CYCLE



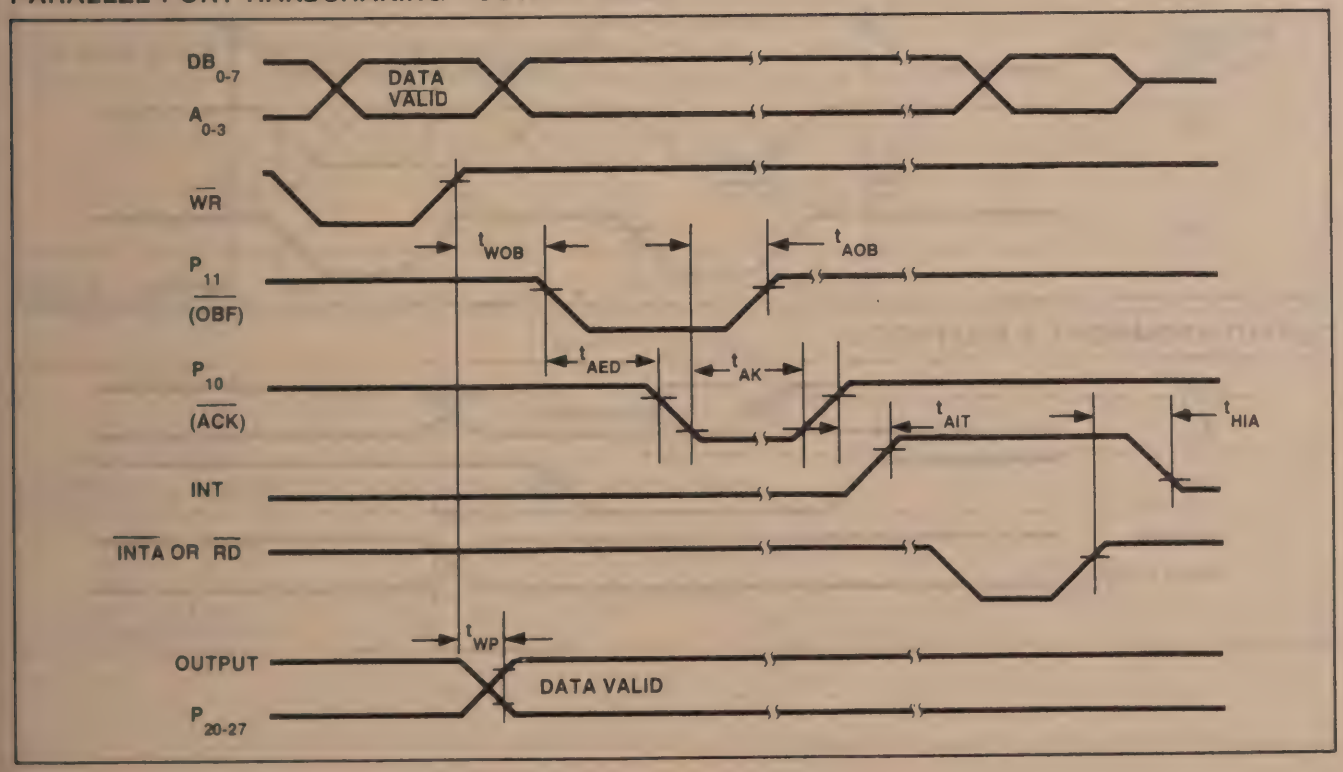
230759-002

WAVEFORMS (Continued)

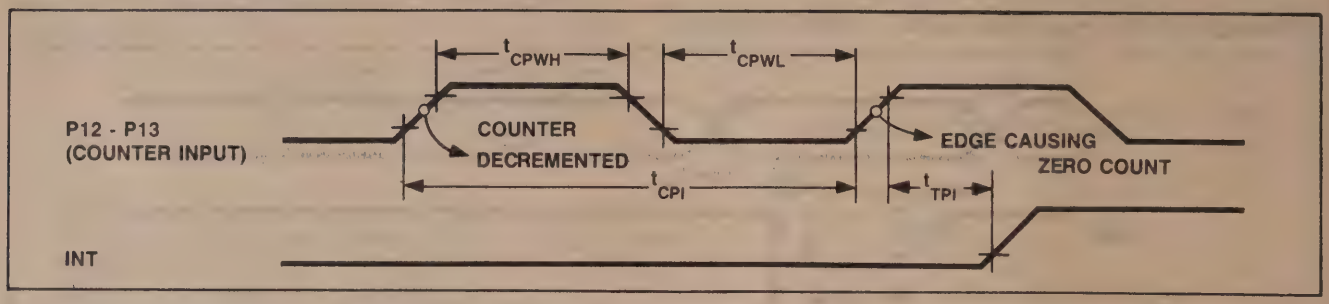
PARALLEL PORT HANDSHAKING - INPUT MODE



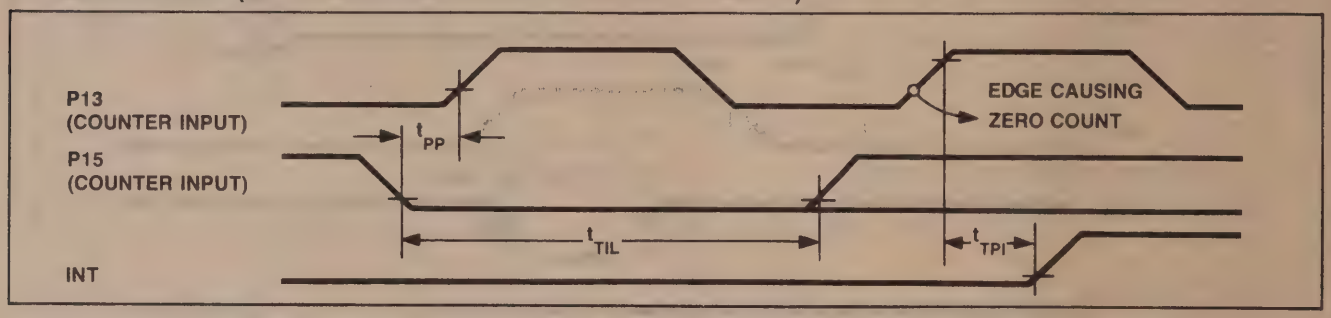
PARALLEL PORT HANDSHAKING - OUTPUT MODE



COUNT PULSE TIMINGS

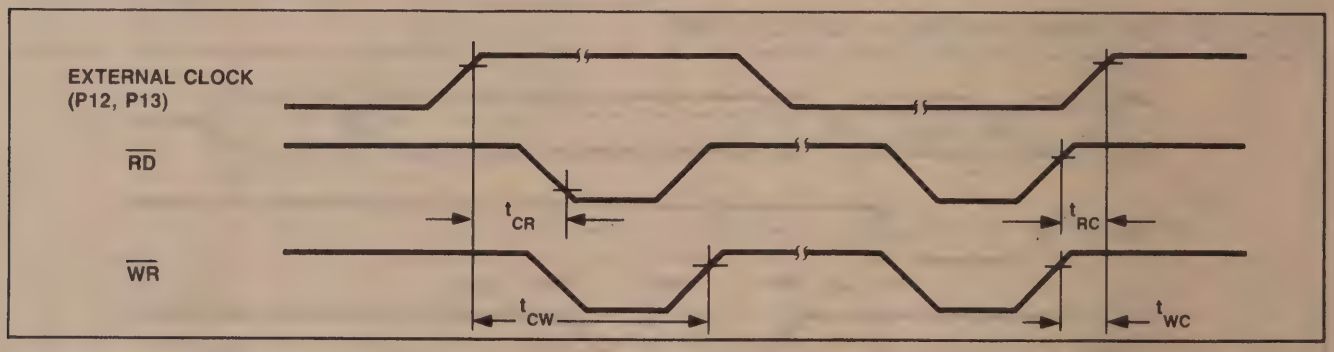


LOADING TIMER (OR CASCADED COUNTER/TIMER 3 AND 5)

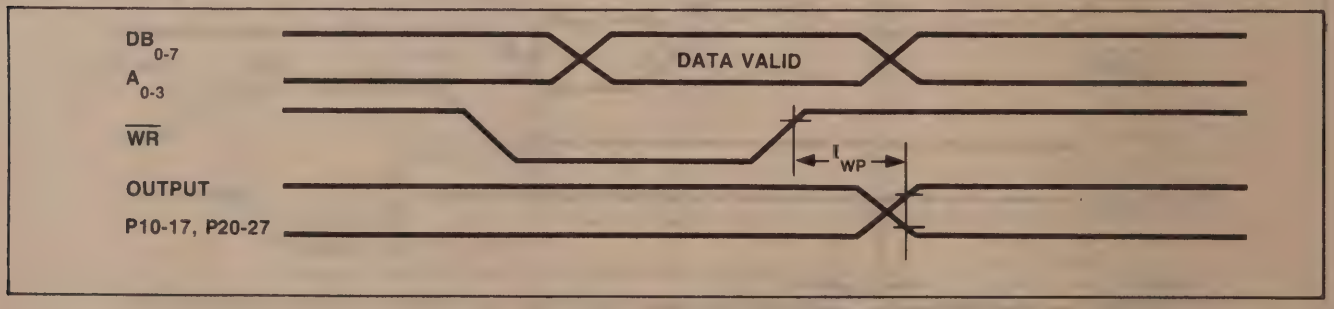


TRIGGER PULSE FOR TIMER 5 (CASCADED EVENT COUNTER/TIMER 3 AND 5)

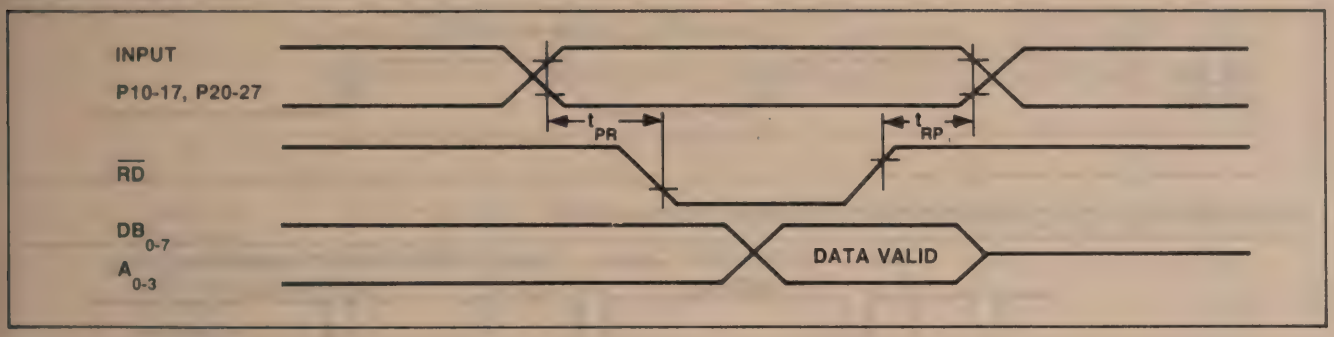
COUNTER TIMER TIMING



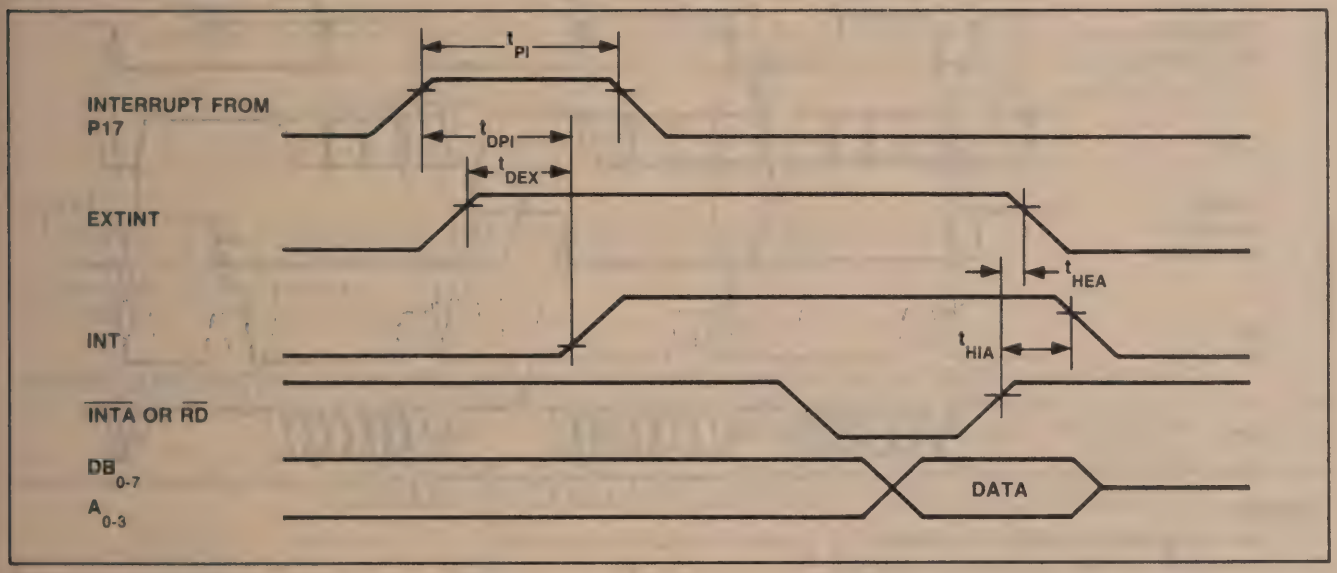
OUTPUT FROM PORT 1 AND PORT 2



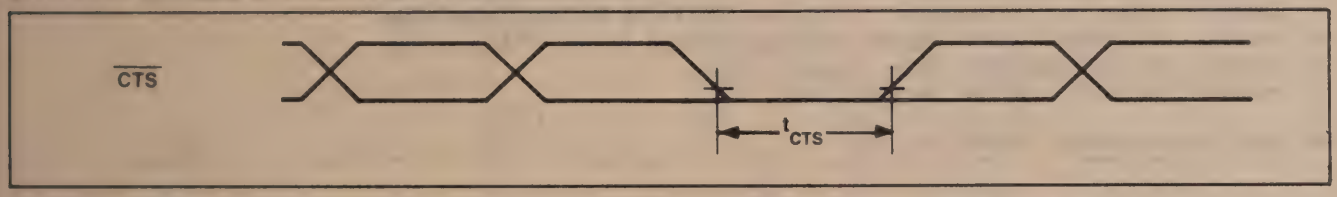
INPUT FROM PORT 1 AND PORT 2



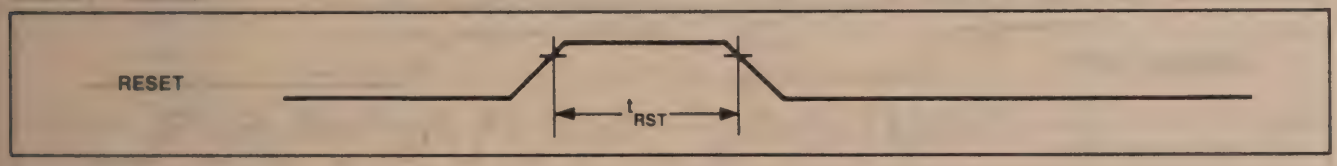
INTERRUPT TIMING



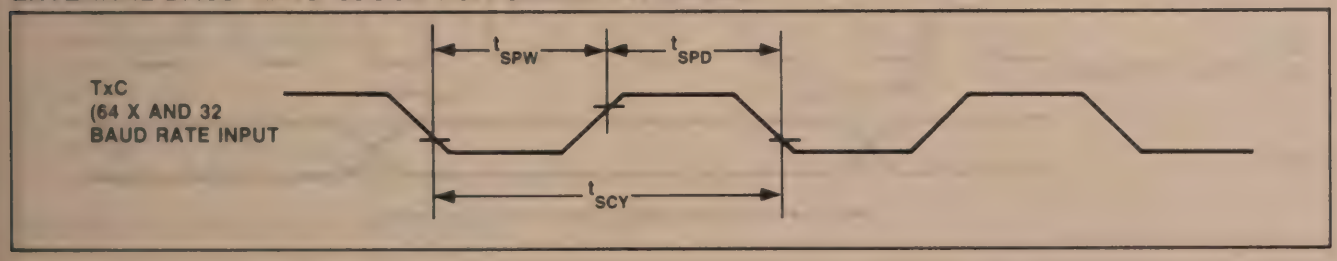
$\overline{CTS}$ FOR SINGLE CHARACTER TRANSMISSION



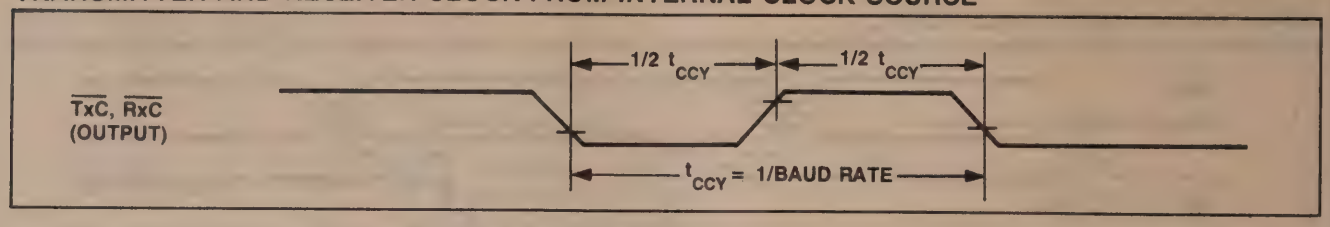
RESET TIMING



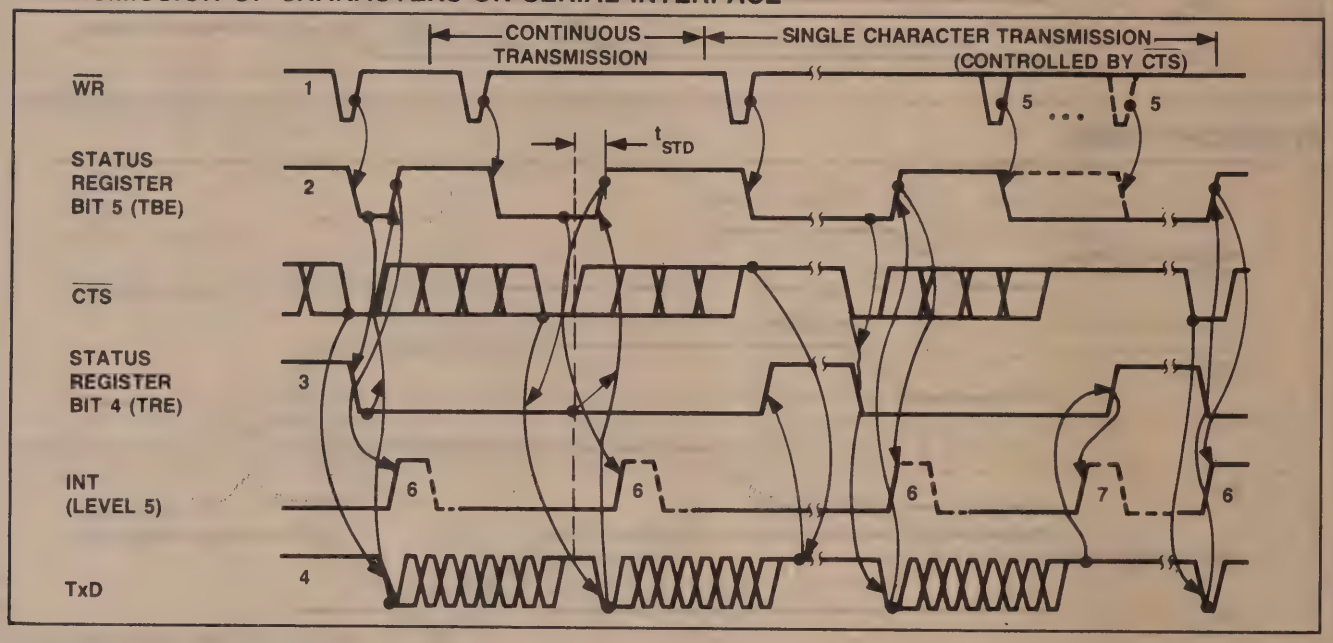
EXTERNAL BAUD RATE CLOCK FOR SERIAL INTERFACE



TRANSMITTER AND RECEIVER CLOCK FROM INTERNAL CLOCK SOURCE



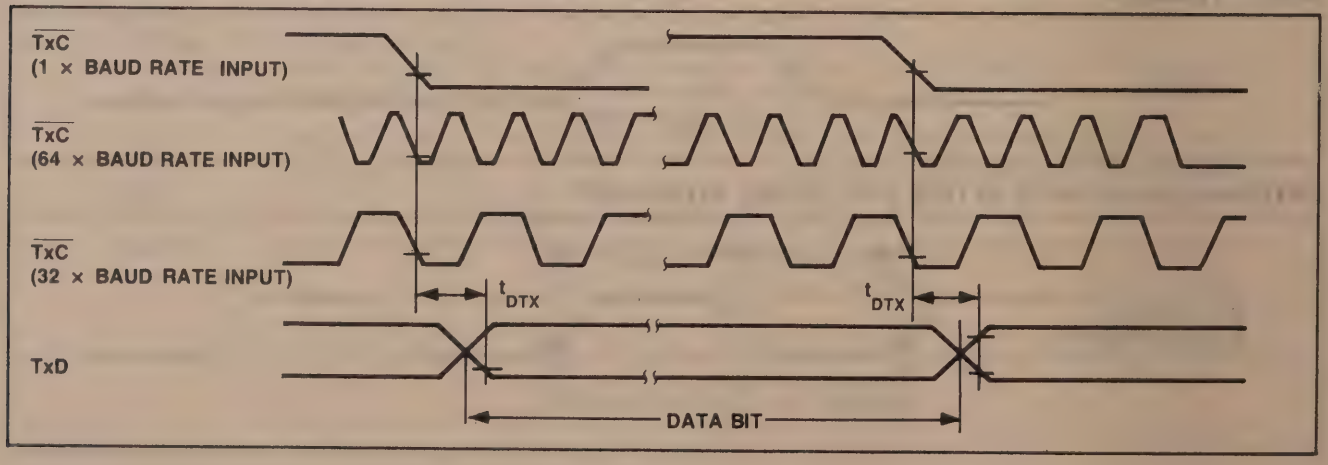
TRANSMISSION OF CHARACTERS ON SERIAL INTERFACE



NOTES:
1. Load transmitter buffer register.
2. Transmitter buffer register is empty.
3. Transmitter register is empty.
4. Character format for this example: 7 Data Bits with Parity Bit and 2 Stop Bits.
5. Loading of transmitter buffer register must be complete before $\overline{CTS}$ goes low.
6. Interrupt due to transmitter buffer register empty.
7. Interrupt due to transmitter register empty.

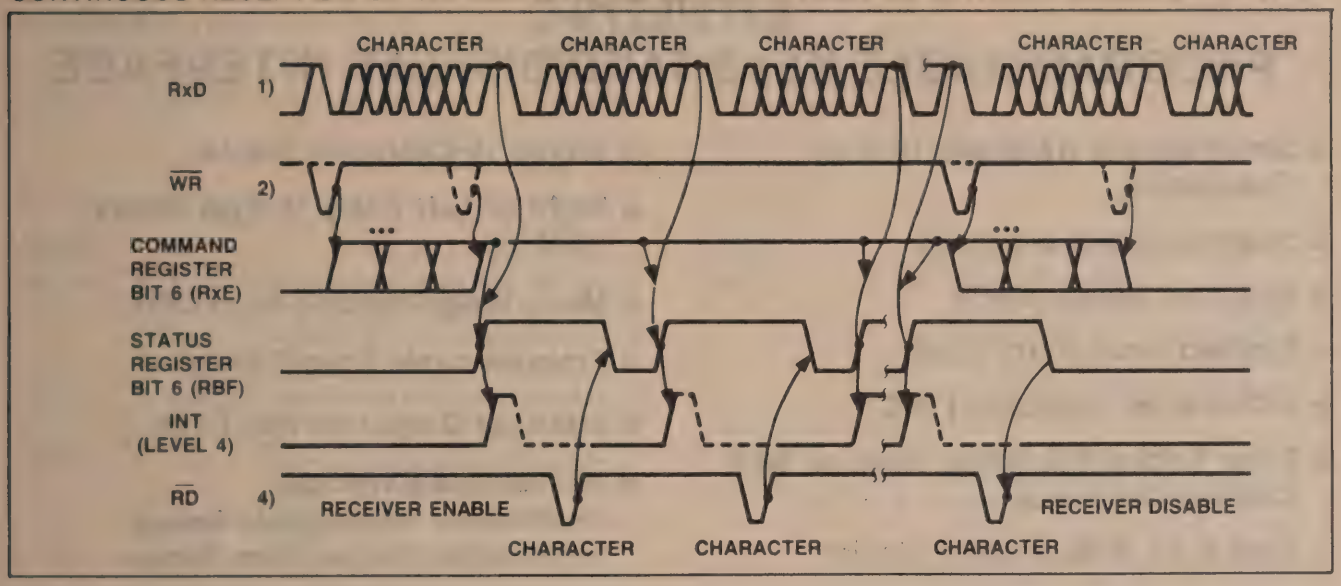
No Status bits are altered when $\overline{RD}$ is active.

DATA BIT OUTPUT ON SERIAL INTERFACE



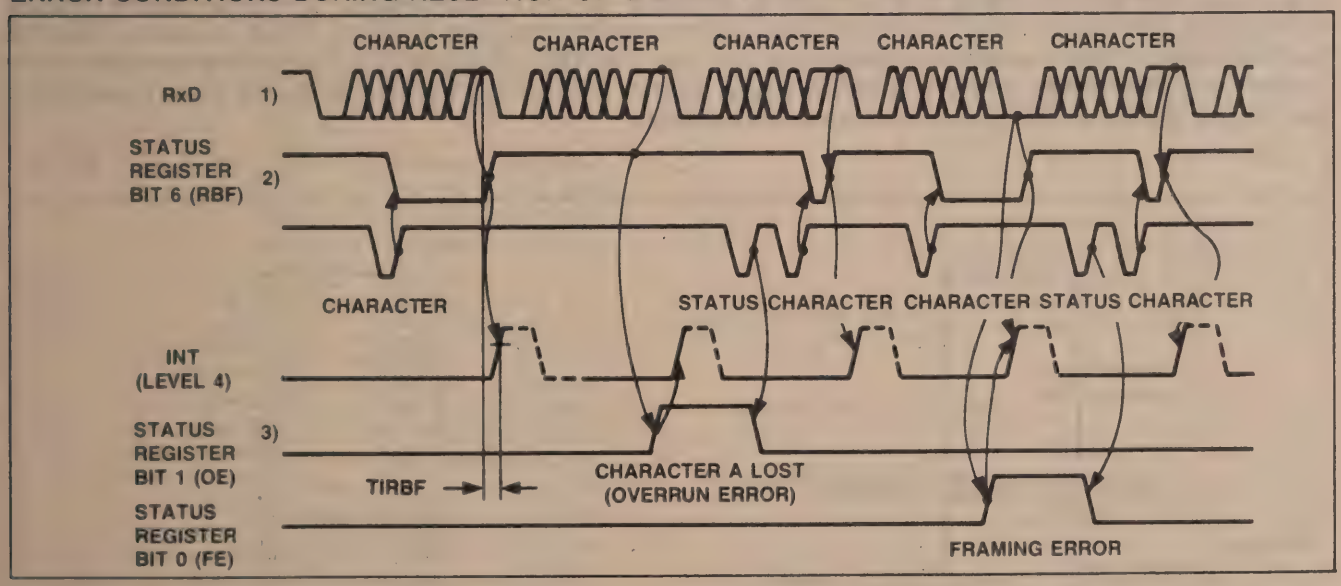
230759-002

CONTINUOUS RECEPTION OF CHARACTERS ON SERIAL INTERFACE WITHOUT ERROR CONDITION



NOTES:
1. Character format for this example: 6 data bits with parity bit and one stop bit.
2. Set or reset bit 6 of command register 3 (enable receiver).
3. Receiver buffer located.
4. Read receiver buffer register.

ERROR CONDITIONS DURING RECEPTION OF CHARACTERS ON THE SERIAL INTERFACE



NOTES:
1. Character format for this example: 6 data bits without parity and one stop bit.
2. Receiver buffer register loaded.
3. Overrun error.
4. Framing error.
5. Interrupt from receiver buffer register loading.
6. Interrupt from overrun error.
7. Interrupt from framing error and loading receiver buffer register.

No status bits are altered when $\overline{RD}$ is active.

8279/8279-5
PROGRAMMABLE KEYBOARD/DISPLAY INTERFACE

- Simultaneous Keyboard Display Operations
- Scanned Keyboard Mode
- Scanned Sensor Mode
- Strobed Input Entry Mode
- 8-Character Keyboard FIFO
- 2-Key Lockout or N-Key Rollover with Contact Debounce
- Dual 8- or 16-Numerical Display
- Single 16-Character Display
- Right or Left Entry 16-Byte Display RAM
- Mode Programmable from CPU
- Programmable Scan Timing
- Interrupt Output on Key Entry
- Available in EXPRESS
  —Standard Temperature Range
  —Extended Temperature Range

The Intel® 8279 is a general purpose programmable keyboard and display I/O interface device designed for use with Intel® microprocessors. The keyboard portion can provide a scanned interface to a 64-contact key matrix. The keyboard portion will also interface to an array of sensors or a strobed interface keyboard, such as the hall effect and ferrite variety. Key depressions can be 2-key lockout or N-key rollover. Keyboard entries are debounced and strobed in an 8-character FIFO. If more than 8 characters are entered, overrun status is set. Key entries set the interrupt output line to the CPU.

The display portion provides a scanned display interface for LED, incandescent, and other popular display technologies. Both numeric and alphanumeric segment displays may be used as well as simple indicators. The 8279 has 16X8 display RAM which can be organized into dual 16X4. The RAM can be loaded or interrogated by the CPU. Both right entry, calculator and left entry typewriter display formats are possible. Both read and write of the display RAM can be done with auto-increment of the display RAM address.

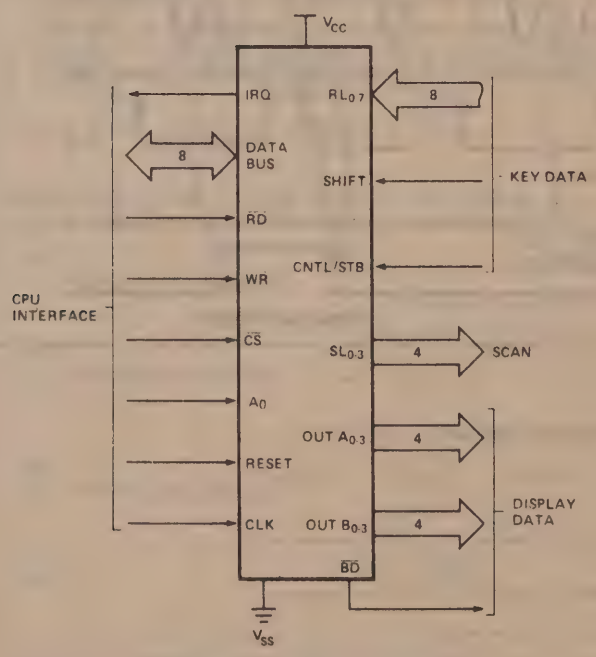


Figure 1. Logic Symbol

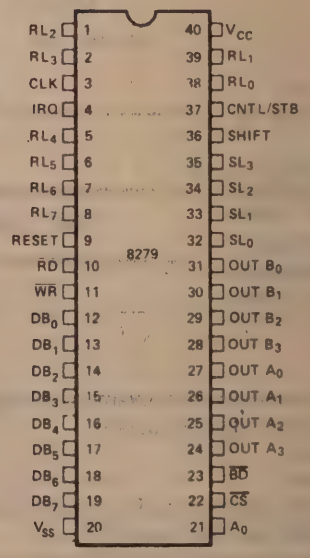


Figure 2. Pin Configuration

HARDWARE DESCRIPTION

The 8279 is packaged in a 40 pin DIP. The following is a functional description of each pin.

Table 1. Pin Descriptions

Symbol	Pin No.	Name and Function
DB_0-DB_7	19-12	**Bi-directional data bus:** All data and commands between the CPU and the 8279 are transmitted on these lines.
CLK	3	**Clock:** Clock from system used to generate internal timing.
RESET	9	**Reset:** A high signal on this pin resets the 8279. After being reset the 8279 is placed in the following mode: 1) 16 8-bit character display —left entry. 2) Encoded scan keyboard—2 key lockout. Along with this the program clock prescaler is set to 31.
CS	22	**Chip Select:** A low on this pin enables the interface functions to receive or transmit.
A_0	21	**Buffer Address:** A high on this line indicates the signals in or out are interpreted as a command or status. A low indicates that they are data.
$\overline{RD}$, $\overline{WR}$	10-11	**Input/Output Read and Write:** These signals enable the data buffers to either send data to the external bus or receive it from the external bus.
IRQ	4	**Interrupt Request:** In a keyboard mode, the interrupt line is high when there is data in the FIFO/Sensor RAM. The interrupt line goes low with each FIFO/Sensor RAM read and returns high if there is still information in the RAM. In a sensor mode, the interrupt line goes high whenever a change in a sensor is detected.
V_{SS}, V_{CC}	20,40	**Ground and power supply pins.**
SL_0-SL_3	32-35	**Scan Lines:** Scan lines which are used to scan the key switch or sensor matrix and the display digits. These lines can be either encoded (1 of 16) or decoded (1 of 4).
RL_0-RL_7	38, 39. 1, 2, 5-8	**Return Line:** Return line inputs which are connected to the scan lines through the keys or sensor switches. They have active internal pullups to keep them high until a switch closure pulls one low. They also serve as an 8-bit input in the Strobed Input mode.
SHIFT	36	**Shift:** The shift input status is stored along with the key position on key closure in the Scanned Keyboard modes. It has an active internal pullup to keep it high until a switch closure pulls it low.
CNTL/STB	37	**Control/Strobed Input Mode:** For keyboard modes this line is used as a control input and stored like status on a key closure. The line is also the strobe line that enters the data into the FIFO in the Strobed Input mode. (Rising Edge). It has an active internal pullup to keep it high until a switch closure pulls it low.
OUT A_0–OUT A_3 OUT B_0–OUT B_3	27-24 31-28	**Outputs:** These two ports are the outputs for the 16 x 4 display refresh registers. The data from these outputs is synchronized to the scan lines (SL_0–SL_3) for multiplexed digit displays. The two 4 bit ports may be blanked independently. These two ports may also be considered as one 8-bit port.
$\overline{BD}$	23	**Blank Display:** This output is used to blank the display during digit switching or by a display blanking command.

FUNCTIONAL DESCRIPTION

Since data input and display are an integral part of many microprocessor designs, the system designer needs an interface that can control these functions without placing a large load on the CPU. The 8279 provides this function for 8-bit microprocessors.

The 8279 has two sections: keyboard and display. The keyboard section can interface to regular typewriter style keyboards or random toggle or thumb switches. The display section drives alphanumeric displays or a bank of indicator lights. Thus the CPU is relieved from scanning the keyboard or refreshing the display.

The 8279 is designed to directly connect to the microprocessor bus. The CPU can program all operating modes for the 8279. These modes include:

Input Modes

- Scanned Keyboard — with encoded (8 x 8 key keyboard) or decoded (4 x 8 key keyboard) scan lines. A key depression generates a 6-bit encoding of key position. Position and shift and control status are stored in the FIFO. Keys are automatically debounced with 2-key lockout or N-key rollover.
- Scanned Sensor Matrix — with encoded (8 x 8 matrix switches) or decoded (4 x 8 matrix switches) scan lines. Key status (open or closed) stored in RAM addressable by CPU.
- Strobed Input — Data on return lines during control line strobe is transferred to FIFO.

Output Modes

- 8 or 16 character multiplexed displays that can be organized as dual 4-bit or single 8-bit ($B_0 = D_0$, $A_3 = D_7$).
- Right entry or left entry display formats.

Other features of the 8279 include:

- Mode programming from the CPU.
- Clock Prescaler
- Interrupt output to signal CPU when there is keyboard or sensor data available.
- An 8 byte FIFO to store keyboard information.
- 16 byte internal Display RAM for display refresh. This RAM can also be read by the CPU.

PRINCIPLES OF OPERATION

The following is a description of the major elements of the 8279 Programmable Keyboard/Display interface device. Refer to the block diagram in Figure 3.

I/O Control and Data Buffers

The I/O control section uses the $\overline{CS}$, A_0, $\overline{RD}$ and $\overline{WR}$ lines to control data flow to and from the various internal registers and buffers. All data flow to and from the 8279 is enabled by $\overline{CS}$. The character of the information, given or desired by the CPU, is identified by A_0. A logic one means the information is a command or status. A logic zero means the information is data. $\overline{RD}$ and $\overline{WR}$ determine the direction of data flow through the Data Buffers. The Data Buffers are bi-directional buffers that connect the internal bus to the external bus. When the chip is not selected ($\overline{CS}$ = 1), the devices are in a high impedance state. The drivers input during $\overline{WR} \bullet \overline{CS}$ and output during $\overline{RD} \bullet \overline{CS}$.

Control and Timing Registers and Timing Control

These registers store the keyboard and display modes and other operating conditions programmed by the CPU. The modes are programmed by presenting the proper command on the data lines with $A_0 = 1$ and then sending a $\overline{WR}$. The command is latched on the rising edge of $\overline{WR}$.

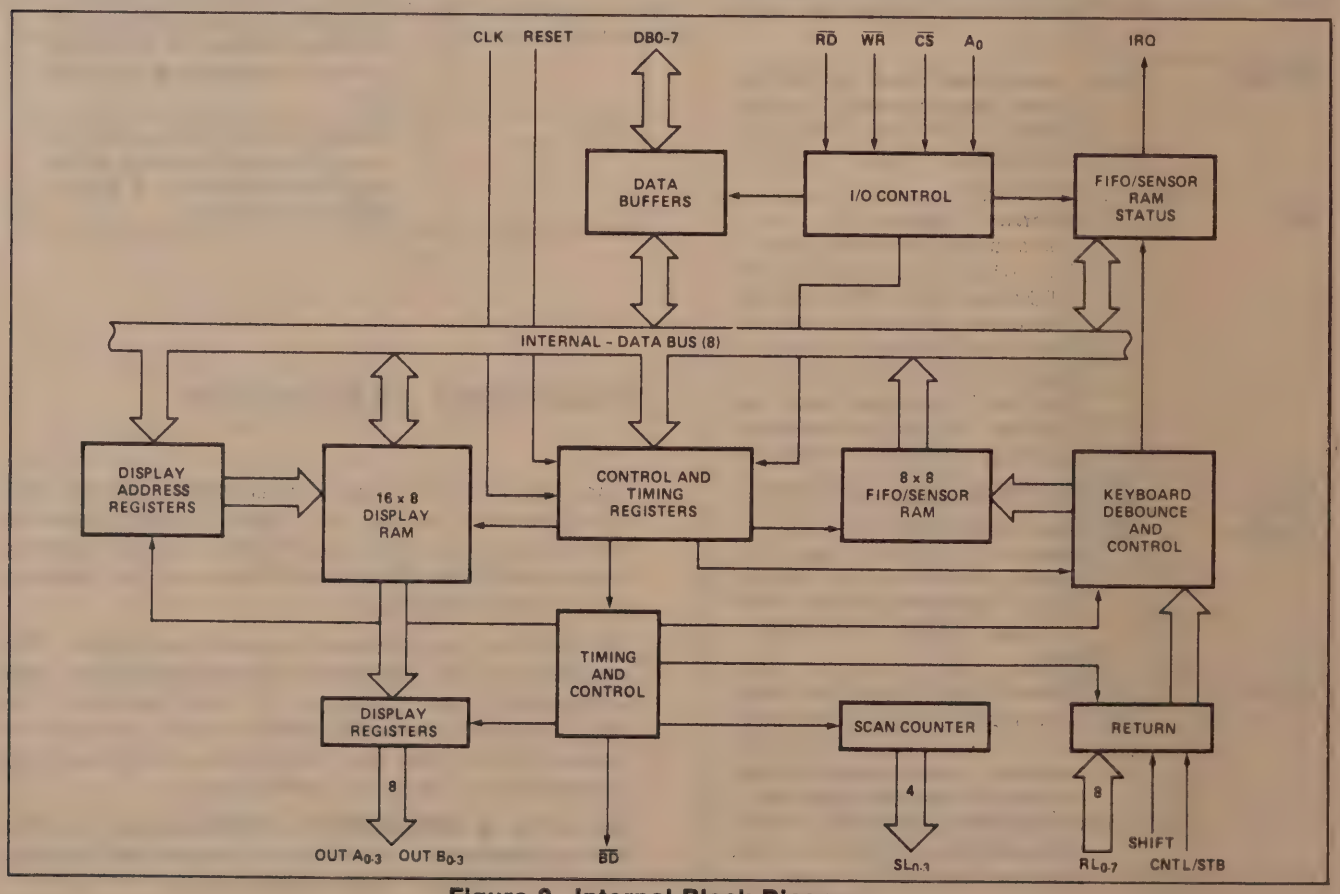


Figure 3. Internal Block Diagram

The command is then decoded and the appropriate function is set. The timing control contains the basic timing counter chain. The first counter is a ÷ N prescaler that can be programmed to yield an internal frequency of 100 kHz which gives a 5.1 ms keyboard scan time and a 10.3 ms debounce time. The other counters divide down the basic internal frequency to provide the proper key scan, row scan, keyboard matrix scan, and display scan times.

Scan Counter

The scan counter has two modes. In the encoded mode, the counter provides a binary count that must be externally decoded to provide the scan lines for the keyboard and display. In the decoded mode, the scan counter decodes the least significant 2 bits and provides a decoded 1 of 4 scan. Note than when the keyboard is in decoded scan, so is the display. This means that only the first 4 characters in the Display RAM are displayed.

In the encoded mode, the scan lines are active high outputs. In the decoded mode, the scan lines are active low outputs.

Return Buffers and Keyboard Debounce and Control

The 8 return lines are buffered and latched by the Return Buffers. In the keyboard mode, these lines are scanned, looking for key closures in that row. If the debounce circuit detects a closed switch, it waits about 10 msec to check if the switch remains closed. If it does, the address of the switch in the matrix plus the status of SHIFT and CONTROL are transferred to the FIFO. In the scanned Sensor Matrix modes, the contents of the return lines is directly transferred to the corresponding row of the Sensor RAM (FIFO) each key scan time. In Strobed Input mode, the contents of the return lines are transferred to the FIFO on the rising edge of the CNTL/STB line pulse.

FIFO/Sensor RAM and Status

This block is a dual function 8 x 8 RAM. In Keyboard or Strobed Input modes, it is a FIFO. Each new entry is written into successive RAM positions and each is then read in order of entry. FIFO status keeps track of the number of characters in the FIFO and whether it is full or empty. Too many reads or writes will be recognized as an error. The status can be read by an $\overline{RD}$ with $\overline{CS}$ low and A_0 high. The status logic also provides an IRQ signal when the FIFO is not empty. In Scanned Sensor Matrix mode, the memory is a Sensor RAM. Each row of the Sensor RAM is loaded with the status of the corresponding row of sensor in the sensor matrix. In this mode, IRQ is high if a change in a sensor is detected.

Display Address Registers and Display RAM

The Display Address Registers hold the address of the word currently being written or read by the CPU and the two 4-bit nibbles being displayed. The read/write addresses are programmed by CPU command. They also can be set to auto increment after each read or write. The Display RAM can be directly read by the CPU after the correct mode and address is set. The addresses for the A and B nibbles are automatically updated by the 8279 to match data entry by the CPU. The A and B nibbles can be entered independently or as one word, according to the mode that is set by the CPU. Data entry to the display can be set to either left or right entry. See Interface Considerations for details.

SOFTWARE OPERATION

8279 commands

The following commands program the 8279 operating modes. The commands are sent on the Data Bus with $\overline{CS}$ low and A_0 high and are loaded to the 8279 on the rising edge of $\overline{WR}$.

Keyboard/Display Mode Set

	MSB							LSB
Code:	0	0	0	D	D	K	K	K

Where DD is the Display Mode and KKK is the Keyboard Mode.

DD

0	0	8 8-bit character display — Left entry
0	1	16 8-bit character display — Left entry*
1	0	8 8-bit character display — Right entry
1	1	16 8-bit character display — Right entry

For description of right and left entry, see Interface Considerations. Note that when decoded scan is set in keyboard mode, the display is reduced to 4 characters independent of display mode set.

KKK

0	0	0	Encoded Scan Keyboard — 2 Key Lockout*
0	0	1	Decoded Scan Keyboard — 2-Key Lockout
0	1	0	Encoded Scan Keyboard — N-Key Rollover
0	1	1	Decoded Scan Keyboard — N-Key Rollover
1	0	0	Encoded Scan Sensor Matrix
1	0	1	Decoded Scan Sensor Matrix
1	1	0	Strobed Input, Encoded Display Scan
1	1	1	Strobed Input, Decoded Display Scan

Program Clock

Code:	0	0	1	P	P	P	P	P

All timing and multiplexing signals for the 8279 are generated by an internal prescaler. This prescaler divides the external clock (pin 3) by a programmable integer. Bits PPPPP determine the value of this integer which ranges from 2 to 31. Choosing a divisor that yields 100 kHz will give the specified scan and debounce times. For instance, if Pin 3 of the 8279 is being clocked by a 2 MHz signal, PPPPP should be set to 10100 to divide the clock by 20 to yield the proper 100 kHz operating frequency.

Read FIFO/Sensor RAM

Code:	0	1	0	AI	X	A	A	A	X = Don't Care

The CPU sets up the 8279 for a read of the FIFO/Sensor RAM by first writing this command. In the Scan Key-

*Default after reset.

board Mode, the Auto-Increment flag (AI) and the RAM address bits (AAA) are irrelevant. The 8279 will automatically drive the data bus for each subsequent read ($A_0 = 0$) in the same sequence in which the data first entered the FIFO. All subsequent reads will be from the FIFO until another command is issued.

In the Sensor Matrix Mode, the RAM address bits AAA select one of the 8 rows of the Sensor RAM. If the AI flag is set (AI = 1), each successive read will be from the subsequent row of the sensor RAM.

Read Display RAM

Code:

0	1	1	AI	A	A	A	A

The CPU sets up the 8279 for a read of the Display RAM by first writing this command. The address bits AAAA select one of the 16 rows of the Display RAM. If the AI flag is set (AI = 1), this row address will be incremented after each following read *or write* to the Display RAM. Since the same counter is used for both reading and writing, this command sets the next read *or write* address and the sense of the Auto-Increment mode for both operations.

Write Display RAM

Code:

1	0	0	AI	A	A	A	A

The CPU sets up the 8279 for a write to the Display RAM by first writing this command. After writing the command with $A_0 = 1$, all subsequent writes with $A_0 = 0$ will be to the Display RAM. The addressing and Auto-Increment functions are identical to those for the Read Display RAM. However, this command does not affect the source of subsequent Data Reads; the CPU will read from whichever RAM (Display or FIFO/Sensor) which was last specified. If, indeed, the Display RAM was last specified, the Write Display RAM will, nevertheless, change the next Read location.

Display Write Inhibit/Blanking

Code:

				A	B	A	B
1	0	1	X	IW	IW	BL	BL

The IW Bits can be used to mask nibble A and nibble B in applications requiring separate 4-bit display ports. By setting the IW flag (IW = 1) for one of the ports, the port becomes marked so that entries to the Display RAM from the CPU do not affect that port. Thus, if each nibble is input to a BCD decoder, the CPU may write a digit to the Display RAM without affecting the other digit being displayed. It is important to note that bit B_0 corresponds to bit D_0 on the CPU bus, and that bit A_3 corresponds to bit D_7.

If the user wishes to blank the display, the BL flags are available for each nibble. The last Clear command issued determines the code to be used as a "blank." This code defaults to all zeros after a reset. Note that both BL flags must be set to blank a display formatted with a single 8-bit port.

Clear

Code:

1	1	0	C_D	C_D	C_D	C_F	C_A

The C_D bits are available in this command to clear all rows of the Display RAM to a selectable blanking code as follows:

C_D	C_D	C_D	
0	X		All Zeros (X = Don't Care)
1	0		AB = Hex 20 (0010 0000)
1	1		All Ones

Enable clear display when = 1 (or by $C_A = 1$)

During the time the Display RAM is being cleared (~160 μs), it may not be written to. The most significant bit of the FIFO status word is set during this time. When the Display RAM becomes available again, it automatically resets.

If the C_F bit is asserted ($C_F = 1$), the FIFO status is cleared and the interrupt output line is reset. Also, the Sensor RAM pointer is set to row 0.

C_A, the Clear All bit, has the combined effect of C_D and C_F; it uses the C_D clearing code on the Display RAM and also clears FIFO status. Furthermore, it resynchronizes the internal timing chain.

End Interrupt/Error Mode Set

Code:

1	1	1	E	X	X	X	X

X = Don't care.

For the sensor matrix modes this command lowers the IRQ line and enables further writing into RAM. (The IRQ line would have been raised upon the detection of a change in a sensor value. This would have also inhibited further writing into the RAM until reset).

For the N-key rollover mode — if the E bit is programmed to "1" the chip will operate in the special Error mode. (For further details, see Interface Considerations Section.)

Status Word

The status word contains the FIFO status, error, and display unavailable signals. This word is read by the CPU when A_0 is high and $\overline{CS}$ and $\overline{RD}$ are low. See Interface Considerations for more detail on status word.

Data Read

Data is read when A_0, $\overline{CS}$ and $\overline{RD}$ are all low. The source of the data is specified by the Read FIFO or Read Display commands. The trailing edge of $\overline{RD}$ will cause the address of the RAM being read to be incremented if the Auto-Increment flag is set. FIFO reads always increment (if no error occurs) independent of AI.

Data Write

Data that is written with A_0, $\overline{CS}$ and $\overline{WR}$ low is always written to the Display RAM. The address is specified by the latest Read Display or Write Display command. Auto-Incrementing on the rising edge of $\overline{WR}$ occurs if AI set by the latest display command.

INTERFACE CONSIDERATIONS

Scanned Keyboard Mode, 2-Key Lockout

There are three possible combinations of conditions that can occur during debounce scanning. When a key is depressed, the debounce logic is set. Other depressed keys are looked for during the next two scans. If none are encountered, it is a single key depression and the key position is entered into the FIFO along with the status of CNTL and SHIFT lines. If the FIFO was empty, IRQ will be set to signal the CPU that there is an entry in the FIFO. If the FIFO was full, the key will not be entered and the error flag will be set. If another closed switch is encountered, no entry to the FIFO can occur. If all other keys are released before this one, then it will be entered to the FIFO. If this key is released before any other, it will be entirely ignored. A key is entered to the FIFO only once per depression, no matter how many keys were pressed along with it or in what order they were released. If two keys are depressed within the debounce cycle, it is a simultaneous depression. Neither key will be recognized until one key remains depressed alone. The last key will be treated as a single key depression.

Scanned Keyboard Mode, N-Key Rollover

With N-key Rollover each key depression is treated independently from all others. When a key is depressed, the debounce circuit waits 2 keyboard scans and then checks to see if the key is still down. If it is, the key is entered into the FIFO. Any number of keys can be depressed and another can be recognized and entered into the FIFO. If a simultaneous depression occurs, the keys are recognized and entered according to the order the keyboard scan found them.

Scanned Keyboard — Special Error Modes

For N-key rollover mode the user can program a special error mode. This is done by the "End Interrupt/Error Mode Set" command. The debounce cycle and key-validity check are as in normal N-key mode. If during a single debounce cycle, two keys are found depressed, this is considered a simultaneous multiple depression, and sets an error flag. This flag will prevent any further writing into the FIFO and will set interrupt (if not yet set). The error flag could be read in this mode by reading the FIFO STATUS word. (See "FIFO STATUS" for further details.) The error flag is reset by sending the normal CLEAR command with $C_F = 1$.

Sensor Matrix Mode

In Sensor Matrix mode, the debounce logic is inhibited. The status of the sensor switch is inputted directly to the Sensor RAM. In this way the Sensor RAM keeps an image of the state of the switches in the sensor matrix. Although debouncing is not provided, this mode has the advantage that the CPU knows how long the sensor was closed and when it was released. A keyboard mode can only indicate a validated closure. To make the software easier, the designer should functionally group the sensors by row since this is the format in which the CPU will read them.

The IRQ line goes high if any sensor value change is detected at the end of a sensor matrix scan. The IRQ line is cleared by the first data read operation if the Auto-Increment flag is set to zero, or by the End Interrupt command if the Auto-Increment flag is set to one.

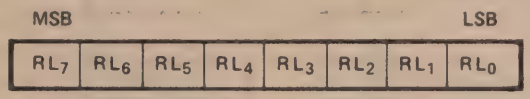
Note: Multiple changes in the matrix Addressed by ($SL_{0-3} = 0$) may cause multiple interrupts. ($SL_0 = 0$ in the Decoded Mode). Reset may cause the 8279 to see multiple changes.

Data Format

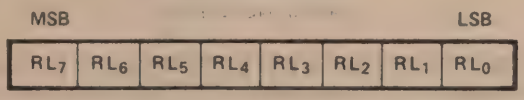
In the Scanned Keyboard mode, the character entered into the FIFO corresponds to the position of the switch in the keyboard plus the status of the CNTL and SHIFT lines (non-inverted). CNTL is the MSB of the character and SHIFT is the next most significant bit. The next three bits are from the scan counter and indicate the row the key was found in. The last three bits are from the column counter and indicate to which return line the key was connected.

SCANNED KEYBOARD DATA FORMAT

In Sensor Matrix mode, the data on the return lines is entered directly in the row of the Sensor RAM that corresponds to the row in the matrix being scanned. Therefore, each switch postion maps directly to a Sensor RAM position. The SHIFT and CNTL inputs are ignored in this mode. Note that switches are not necessarily the only thing that can be connected to the return lines in this mode. Any logic that can be triggered by the scan lines can enter data to the return line inputs. Eight multiplexed input ports could be tied to the return lines and scanned by the 8279.



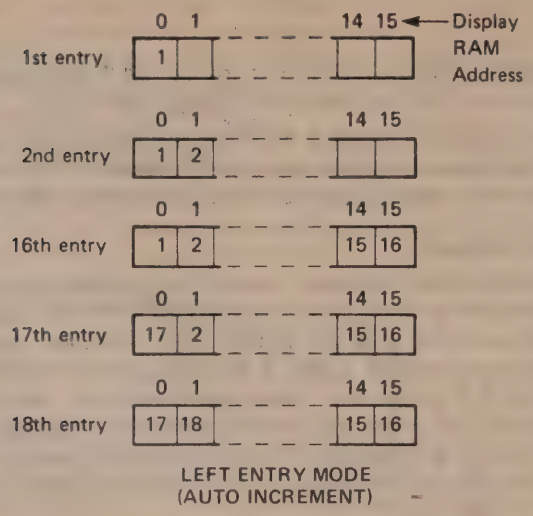
In Strobed Input mode, the data is also entered to the FIFO from the return lines. The data is entered by the rising edge of a CNTL/STB line pulse. Data can come from another encoded keyboard or simple switch matrix. The return lines can also be used as a general purpose strobed input.



Display

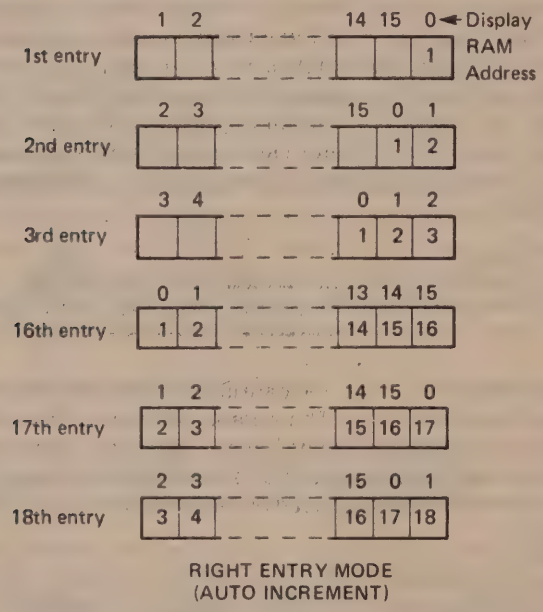
Left Entry

Left Entry mode is the simplest display format in that each display position directly corresponds to a byte (or nibble) in the Display RAM. Address 0 in the RAM is the left-most display character and address 15 (or address 7 in 8 character display) is the right most display character. Entering characters from position zero causes the display to fill from the left. The 17th (9th) character is entered back in the left most position and filling again proceeds from there.



Right Entry

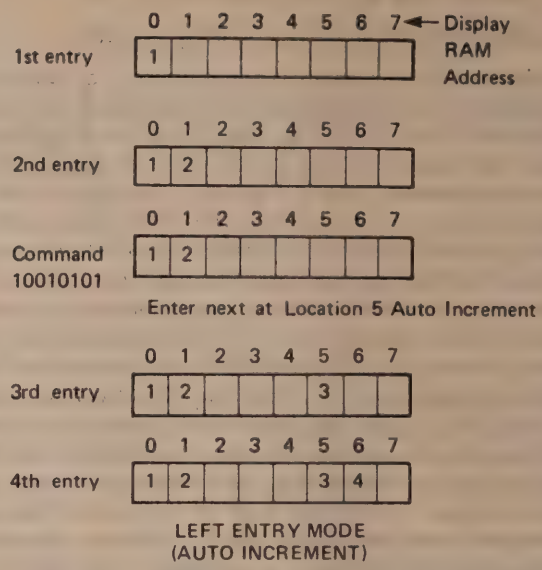
Right entry is the method used by most electronic calculators. The first entry is placed in the right most display character. The next entry is also placed in the right most character after the display is shifted left one character. The left most character is shifted off the end and is lost.



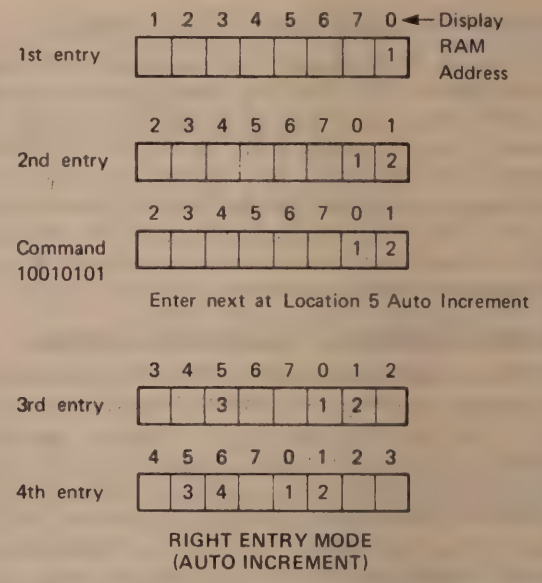
Note that now the display position and register address do not correspond. Consequently, entering a character to an arbitrary position in the Auto Increment mode may have unexpected results. Entry starting at Display RAM address 0 with sequential entry is recommended.

Auto Increment

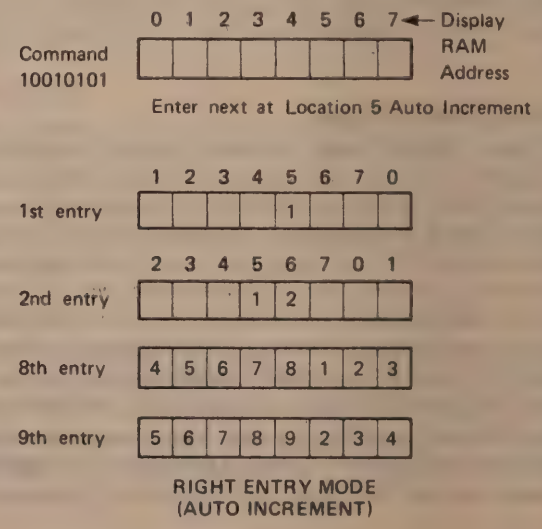
In the Left Entry mode, Auto Incrementing causes the address where the CPU will next write to be incremented by one and the character appears in the next location. With non-Auto Incrementing the entry is both to the same RAM address and display position. Entry to an arbitrary address in the Auto Increment mode has no undesirable side effects and the result is predictable:



In the Right Entry mode, Auto Incrementing and non Incrementing have the same effect as in the Left Entry except if the address sequence is interrupted:



Starting at an arbitrary location operates as shown below:



Entry appears to be from the initial entry point.

8/16 Character Display Formats

If the display mode is set to an 8 character display, the on duty-cycle is double what it would be for a 16 character display (e.g., 5.1 ms scan time for 8 characters vs. 10.3 ms for 16 characters with 100 kHz internal frequency).

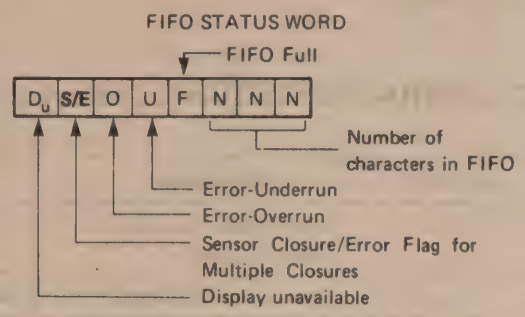
G. FIFO Status

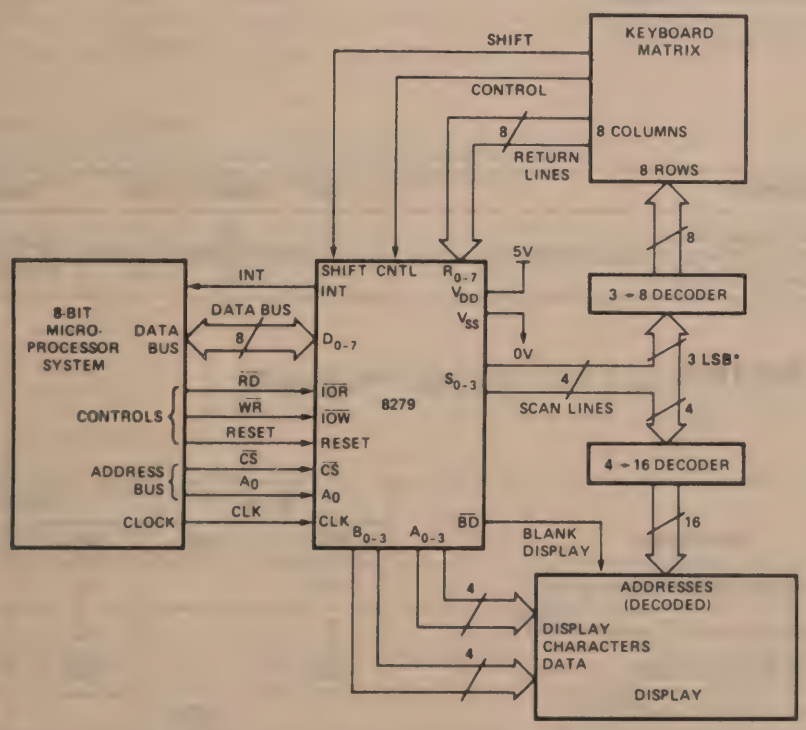
FIFO status is used in the Keyboard and Strobed Input modes to indicate the number of characters in the FIFO and to indicate whether an error has occurred. There are two types of errors possible: overrun and underrun. Overrun occurs when the entry of another character into a full FIFO is attempted. Underrun occurs when the CPU tries to read an empty FIFO.

The FIFO status word also has a bit to indicate that the Display RAM was unavailable because a Clear Display or Clear All command had not completed its clearing operation.

In a Sensor Matrix mode, a bit is set in the FIFO status word to indicate that at least one sensor closure indication is contained in the Sensor RAM.

In Special Error Mode the S/E bit is showing the error flag and serves as an indication to whether a simultaneous multiple closure error has occurred.





*Do not drive the keyboard decoder with the MSB of the scan lines.

Figure 4. System Block Diagram

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature . . . . . 0°C to 70°C
Storage Temperature . . . . . -65°C to 125°C
Voltage on any Pin with
Respect to Ground . . . . . -0.5V to +7V
Power Dissipation . . . . . 1 Watt

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS [T_A = 0°C to 70°C, V_{SS} = 0V, (NOTE 3)]*

Symbol	Parameter	Min.	Max.	Unit	Test Conditions
V_{IL1}	Input Low Voltage for Return Lines	-0.5	1.4	V	
V_{IL2}	Input Low Voltage for All Others	-0.5	0.8	V	
V_{IH1}	Input High Voltage for Return Lines	2.2		V	
V_{IH2}	Input High Voltage for All Others	2.0		V	
V_{OL}	Output Low Voltage		0.45	V	Note 1
V_{OH1}	Output High Voltage on Interrupt Line	3.5		V	Note 2
V_{OH2}	Other Outputs	2.4			I_{OH} = −400 μA 8279-5 −100 μA 8279
I_{IL1}	Input Current on Shift, Control and Return Lines		+10 -100	μA μA	$V_{IN} = V_{CC}$ V_{IN} = 0V
I_{IL2}	Input Leakage Current on All Others		±10	μA	$V_{IN} = V_{CC}$ to 0V
I_{OFL}	Output Float Leakage		±10	μA	$V_{OUT} = V_{CC}$ to 0.45V
I_{CC}	Power Supply Current		120	mA	

CAPACITANCE

Symbol	Parameter	Typ.	Max.	Unit	Test Conditions
C_{IN}	Input Capacitance	5	10	pF	f_C = 1 MHz Unmeasured pins returned to V_{SS}
C_{OUT}	Output Capacitance	10	20	pF	

A.C. CHARACTERISTICS [T_A = 0°C to 70°C, V_{SS} = 0V, (Note 3)] *

Bus Parameters

READ CYCLE

Symbol	Parameter	8279		8279-5		Unit
		Min.	Max.	Min.	Max.	
t_{AR}	Address Stable Before $\overline{READ}$	50		0		ns
t_{RA}	Address Hold Time for $\overline{READ}$	5		0		ns
t_{RR}	$\overline{READ}$ Pulse Width	420		250		ns
t_{RD} [4]	Data Delay from $\overline{READ}$		300		150	ns
t_{AD} [4]	Address to Data Valid		450		250	ns
t_{DF}	$\overline{READ}$ to Data Floating	10	100	10	100	ns
t_{RCY}	Read Cycle Time	1		1		μs

A.C. CHARACTERISTICS (Continued)

WRITE CYCLE

Symbol	Parameter	8279 Min.	8279 Max.	8279-5 Min.	8279-5 Max.	Unit
t_{AW}	Address Stable Before $\overline{WRITE}$	50		0		ns
t_{WA}	Address Hold Time for $\overline{WRITE}$	20		0		ns
t_{WW}	$\overline{WRITE}$ Pulse Width	400		250		ns
t_{DW}	Data Set Up Time for $\overline{WRITE}$	300		150		ns
t_{WD}	Data Hold Time for $\overline{WRITE}$	40		0		ns
t_{WCY}	Write Cycle Time	1		1		μs

OTHER TIMINGS

Symbol	Parameter	8279 Min.	8279 Max.	8279-5 Min.	8279-5 Max.	Unit
$t_{\phi W}$	Clock Pulse Width	230		120		nsec
t_{CY}	Clock Period	500		320		nsec

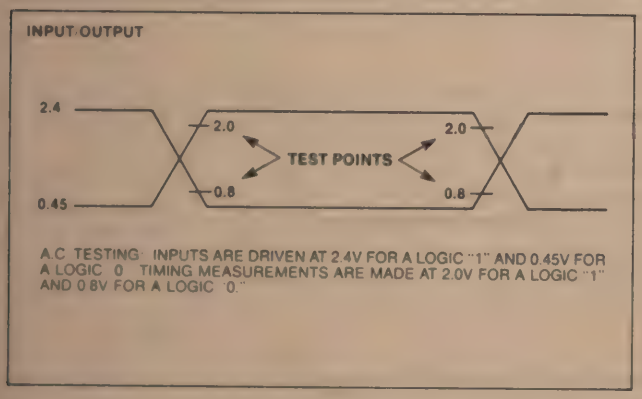
Keyboard Scan Time ..... 5.1 msec
Keyboard Debounce Time ..... 10.3 msec
Key Scan Time ..... 80 μsec
Display Scan Time ..... 10.3 msec

Digit-on Time ..... 480 μsec
Blanking Time ..... 160 μsec
Internal Clock Cycle[5] ..... 10 μsec

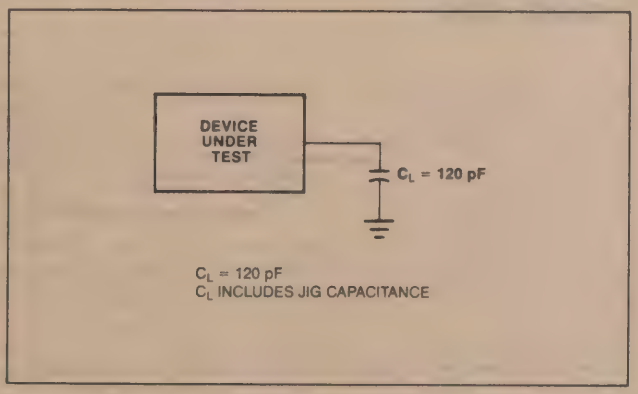
NOTES:
1. 8279, I_{OL} = 1.6mA; 8279-5, I_{OL} = 2.2mA.
2. I_{OH} = −100 μA
3. 8279, V_{CC} = +5V ±5%; 8279-5, V_{CC} = +5V ±10%.
4. 8279, C_L = 100pF; 8279-5, C_L = 150pF.
5. The Prescaler should be programmed to provide a 10 μs internal clock cycle.

* For Extended Temperature EXPRESS, use M8279A electrical parameters.

A.C. TESTING INPUT, OUTPUT WAVEFORM

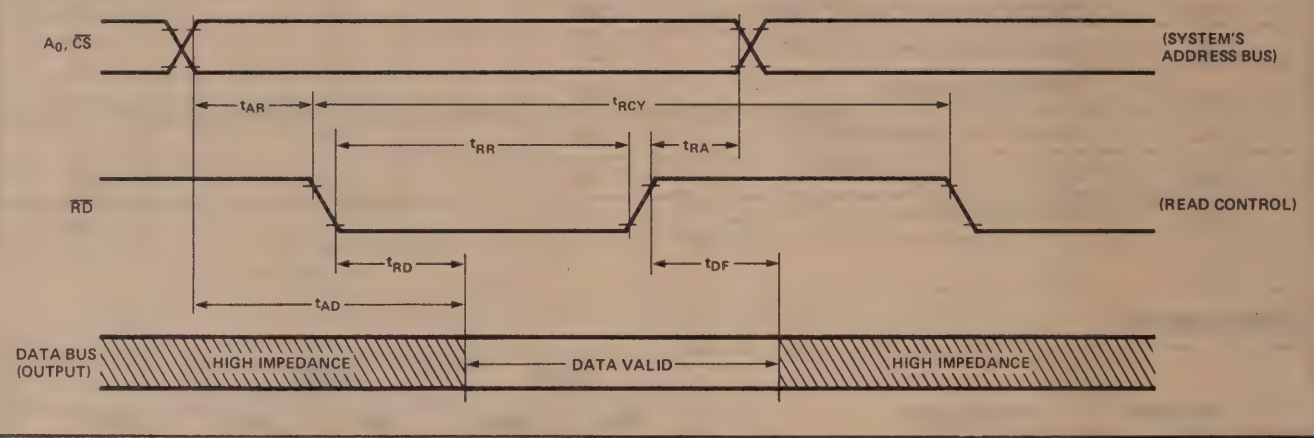


A.C. TESTING LOAD CIRCUIT

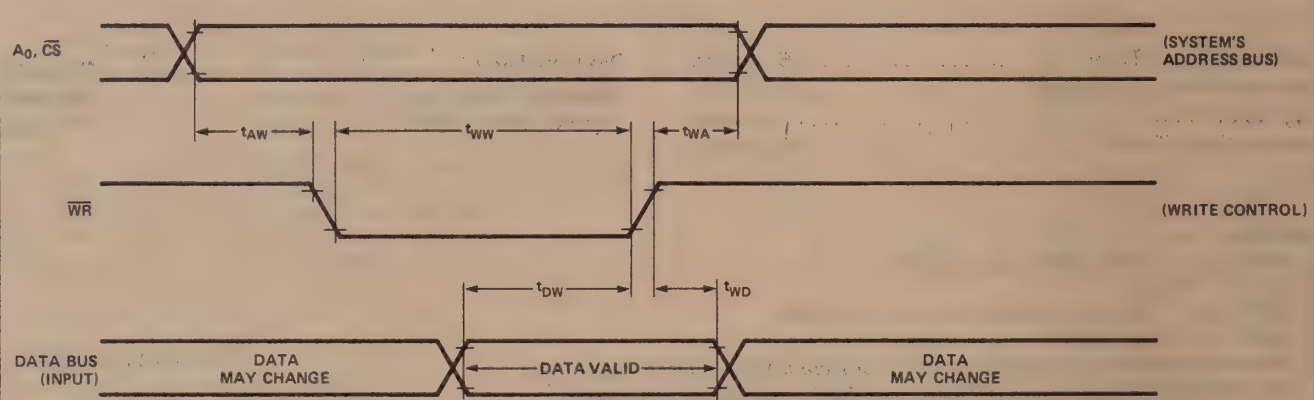


WAVEFORMS

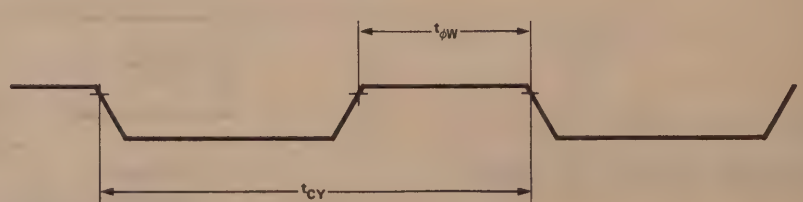
READ OPERATION



WRITE OPERATION



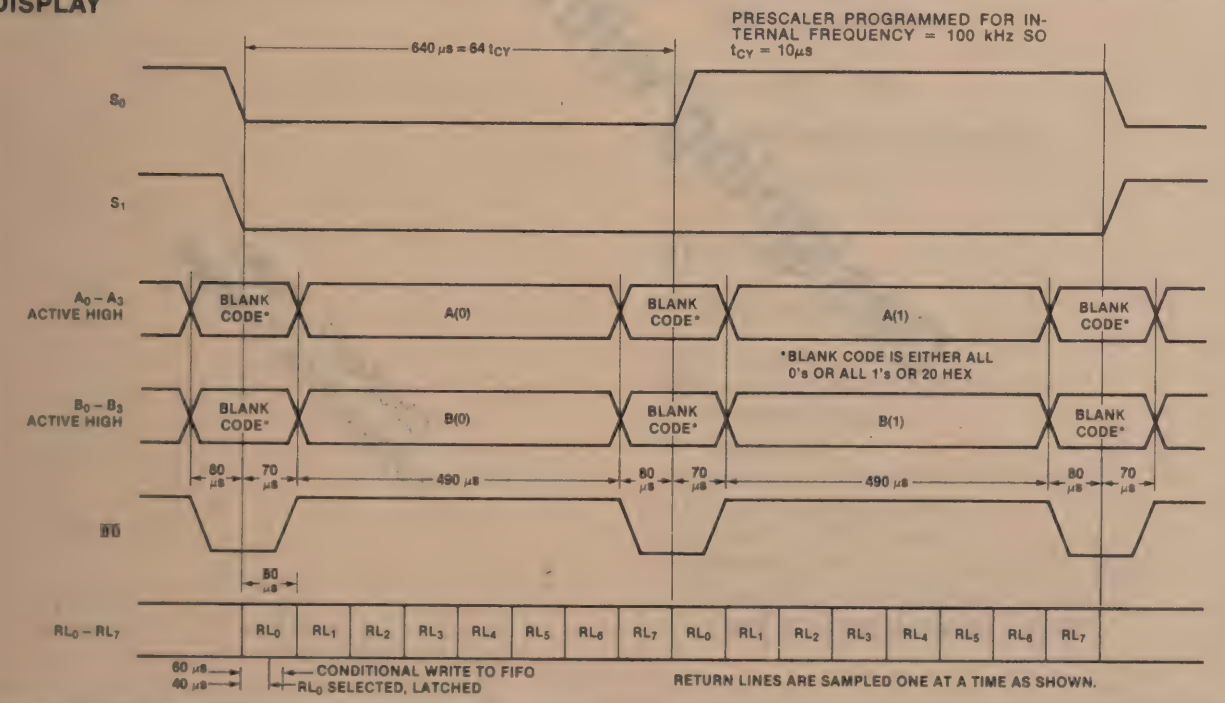
CLOCK INPUT



WAVEFORMS (Continued)

SCAN

DISPLAY



NOTE: SHOWN IS ENCODED SCAN LEFT ENTRY
S_2-S_3 ARE NOT SHOWN BUT THEY ARE SIMPLY S_1 DIVIDED BY 2 AND 4

intel

APPLICATION NOTE

AP-153

June 1983

Designing with the 8256

Charles T. Yager
Applications Engineer

210907-002

Designing with the 8256

Contents

INTRODUCTION

DESCRIPTION OF THE MUART

- Microprocessor Bus Interface
- Command and Status Registers
- Clock Circuitry
  - System Clock Prescaler
  - Timer Prescaler
- Asynchronous Serial Interface
  - Receiver Section of the UART
  - Receive Break Detect
  - Transmitter Section of the UART
  - Transmit Break Features
  - Modification Register
- Parallel I/O
  - Two Wire Byte Handshake
- Event Counter/Timers
- Interrupt Controller
  - MCS-85/8256 Interrupt Operation
  - MCS-86/88/8256 Interrupt Operation
  - Using the Interrupt Controller Without INTA
  - Interrupt Registers
  - Interrupt Modes
  - Edge Triggering
  - Level Triggering
  - Cascading the MUART's Interrupt Controller
  - Polling the MUART

PIN DESCRIPTIONS

DESCRIPTION OF REGISTERS

- Hardware Reset

INTERFACING

PROGRAMMING

- Initialization
- Operating the Serial Interface
  - Transmitting
  - Receiving
- Operating the Parallel Interface
  - Loading Port 1 and 2
  - Reading Port 1 and 2
- Operating the Event Counter/Timers
  - Loading Event Counter/Timers
  - Reading Event Counter/Timers

Contents (cont.)

APPLICATION EXAMPLE

Description of the Line Printer Multiplexer
Description of the Hardware
Description of the Software
Buffer Management
Using the LPM with the MDS SERIES II OR SERIES III

APPENDIX

Listing of the Line Printer Multiplexer Software
Listing of the WRITE Program
MUART Registers

INTRODUCTION

The INTEL 8256 MUART is a Multifunction Universal Asynchronous Receiver Transmitter designed to be used for serial asynchronous communication while also providing hardware support for parallel I/O, timing, counting and interrupt control. Its versatile design allows it to be directly connected to the MCS®-85, iAPX-86, iAPX-88, iAPX-186, and iAPX-188 microcomputer systems plus the MCS-48 and MCS-51 family of single-chip microcomputers.

The four commonly used peripheral functions contained in the MUART are:

1) Full-duplex, double-buffered serial asynchronous Receiver/Transmitter with an on-chip Baud Rate Generator
2) Two – 8-bit parallel I/O ports
3) Five – 8-bit counters/timers
4) 8-level priority interrupt controller

This manual can be divided into two parts. The first part describes the MUART in detail, including its functions, registers and pins. This section also describes the interface between the MUART and Intel CPUs plus a discussion on programming considerations. The second section provides an application example: a MUART-based line printer multiplexer. The Appendix contains software listings for the line printer multiplexer and some useful reference information.

DESCRIPTION OF THE MUART

The MUART can be logically partitioned into seven sections: the microprocessor bus interface, the command and status registers, clocking circuitry, asynchronous serial communication, parallel I/O, timer/event counters, and the interrupt controller. This can be seen from the block diagram of the 8256 MUART as shown in Figure 1. The MUART's pin configuration can be seen in Figure 2.

Microprocessor Bus Interface

The microprocessor bus interface is the hardware section of the MUART which allows a μP to communicate with the MUART. It consists of tristate bi-directional data-bus buffers, an address latch, a chip select ($\overline{CS}$) latch and bus control logic. In order to provide all of the MUART's functions in a 40-pin DIP while retaining direct register addressing, a multiplexed address/data bus is used.

Address/Data Bus

The MUART contains 16 internal directly addressable read/write registers. Four of the eight address/data lines are used to generate the address. When using 8-bit microprocessors such as MCS-85, MCS-48 and MCS-51, AD0 – AD3 are used to address the 16 internal registers while Address/Data line 4 (AD4) is not used for addressing. For 16-bit systems, AD1 – AD4 are used to generate the address for the internal data registers and AD0 is used as a second active low chip select.

$\overline{RD}$, $\overline{WR}$, $\overline{CS}$

The 8256 bus interface uses the standard bus control signals which are compatible with all Intel peripherals and microprocessors. The chip select signal ($\overline{CS}$), typically derived from an address decoder, is latched along with the address on the falling edge of ALE. As a result, chip select does not have to remain low for the entire bus cycle. However, the data bus buffers will remain tristated unless an $\overline{RD}$ or a $\overline{WR}$ signal becomes active while chip select has been latched in low.

INT, $\overline{INTA}$

The INT and $\overline{INTA}$ signals are used to interrupt the CPU and receive the CPU's acknowledgment to the interrupt request. The MUART can vector the CPU to the appropriate service routine depending on the source of the interrupt.

RESET

When a high level occurs on the RESET pin, the MUART is placed in a known initial state. This initial state is described under "Hardware Reset."

Command and Status Register

There are three command registers and one status register as shown in Figure 1. The three command registers are read/write registers while the status register is a read only. The command registers configure the MUART for its operating environment (i.e., 8 or 16 bits CPU, system clock frequency). In addition, they direct its higher level functions such as controlling the UART, selecting modes of operation for the interrupt controller, and choosing the fundamental frequency for the timers. Command Register 3 is the only register in the MUART which is a bit set/reset register, allowing the programmer to simply perform one write to set or reset any of the bits.

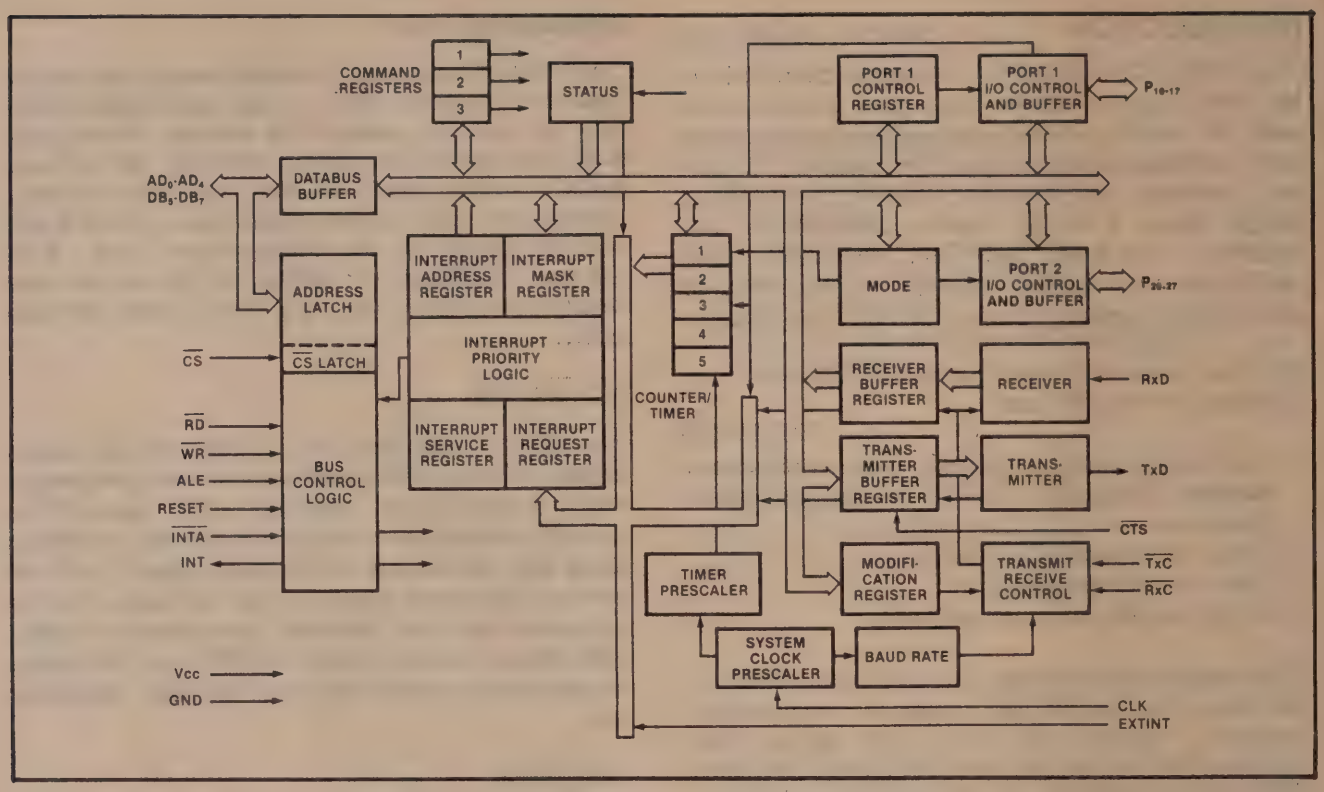


Figure 1. Block Diagram of the 8256 MUART

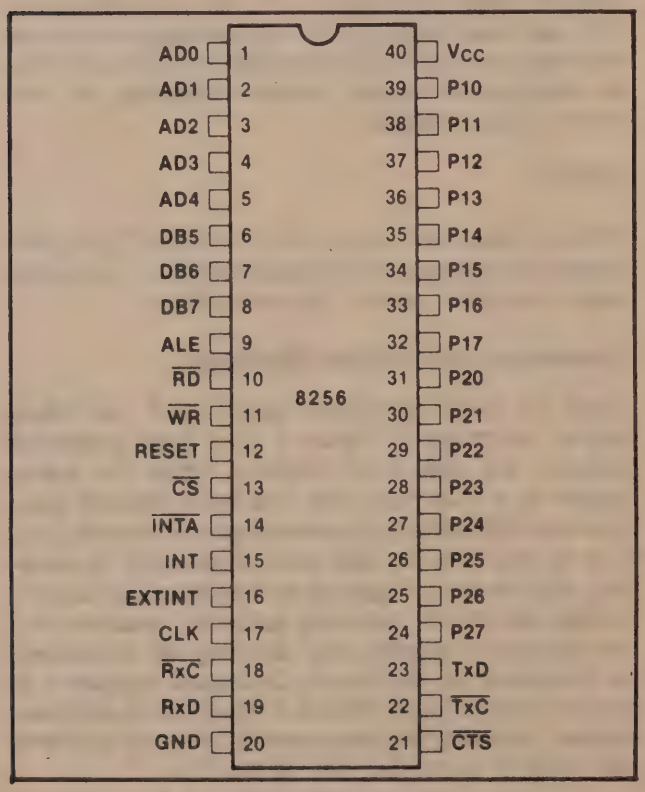


Figure 2. MUART Pin Configuration

The status register provides all of the information about the status of the UART's transmitter and receiver as well as the status of the interrupt pin. The status register is the only read only register in the MUART.

CLOCK CIRCUITRY

The clock for the five timers and baud rate generator is derived from the system clock. The system clock, pin 17 (CLK), is fed into a system clock prescaler which in turn feeds the five timers and the baud rate generator. The MUART's system clock can be asynchronous to the microprocessor's clock.

System Clock Prescaler

The system clock prescaler is a programmable divider which normalizes the internal clocking frequency for the timers and baud rate generator to 1.024MHz. It divides the system clock (CLK) by 1, 2, 3, or 5, allowing clock frequencies of 1.024MHz, 2.048MHz, 3.072MHz or 5.12MHz. (The commonly used 6.144MHz crystal frequency for the 8085 results in a 3.072MHz frequency from the 8085's CLK pin.) If the system clock is not one of the four frequencies mentioned above, then the frequency of the baud rate generator and the timers will be nonstandard;

however, the MUART will still run as long as the system clock meets the data sheet tcy spec.

Timer Prescaler

The timer prescaler permits the user to select one of two fundamental timing frequencies for all of the MUART's timers, either 1KHz or 16KHz. The frequency selection is made via Command Register 0.

Asynchronous Serial Interface

The asynchronous serial interface of the MUART is a full-duplex double-buffered transmitter and receiver with separate control registers. The standard asynchronous format is used as shown in Figure 3. The operation of the UART section of the MUART is very similar to the operation of the 8251A USART.

Receiver Section of the UART

The serial asynchronous receiver section contains a serial shift register, a receiver buffer register and receiver control logic. The serial input data is clocked into the receive shift register from the RxD pin at the specified baud rate. The sampling actually takes place at the rising edge of RxC, assuming an external clock,

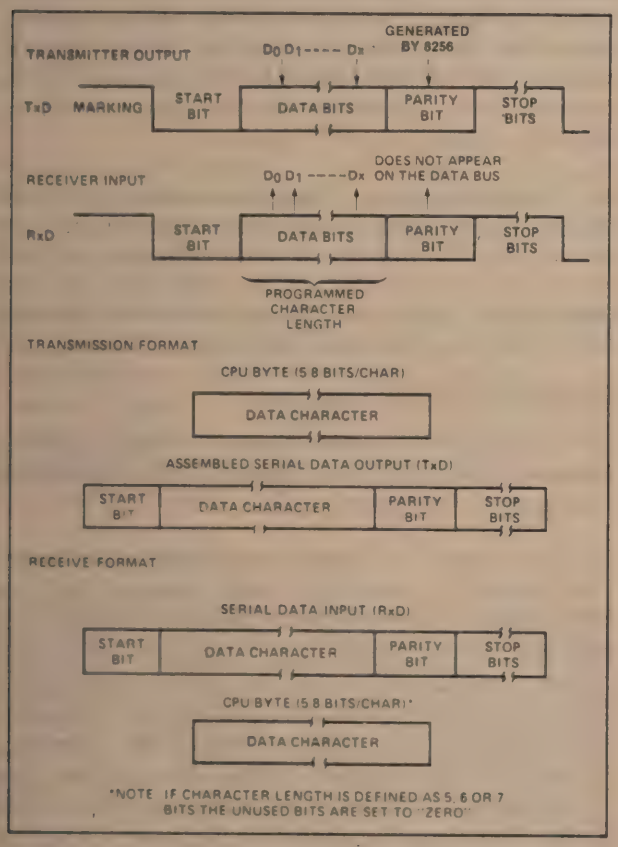


Figure 3. Asynchronous Format

or at the rising edge of the internal baud clock. When the receiver is enabled but inactive, the receive logic is sampling RxD at either 32 or 64 times the bit rate, looking for a change from the Mark (high) to the Space (low) state. This is commonly referred to as the start bit search mode. When this state change occurs, the receive logic waits one half of a bit time and then samples RxD again. If RxD is still in the Space state, the receive logic begins to clock in the receive data beginning one bit period later. If RxD has returned to the Mark state (i.e., false start bit), the receive logic will return to the start bit search mode.

Normally the received data is sampled in the center of each bit, however it is possible to adjust the location where the bit is sampled. This feature is controlled by the modification register.

The bit rate of the serial receive data is derived from either the internal baud rate generator or an external clock. When using an external clock, the programmer has a choice of three sampling rates: 1x, 32x, or 64x, Using the internal baud rate generator, the sampling rates are all 64x except for 19.2 Kbps which is 32x.

When the serial shift register clocks in the stop bit, an internal load pulse is generated which transfers the contents of the shift register into the receive buffer. This transfer takes place during the first half of the first stop bit. The load pulse also triggers several other signals relevant to the receive section including Receive Buffer Full (RBF), Parity Error (PE), Overrun Error (OE), and Framing Error (FE). These four status bits are updated after the middle of the first stop bit when the receive buffer has already been latched. Each one of these four status bits are latched. They are reset on the rising edge of the first read pulse (RD) addressed to the status register. A complete description of the status register is given in the section "Description of the Registers."

When the serial receiver is disabled (via bit 6 of Command Register 3) the load pulse is suppressed. The result is that the receive buffer is not loaded with the contents of the shift register, and the RBF, PE, OE, and FE bits in the status register are not updated. Even though the receiver is disabled, the serial shift register will still be clocking in the data from RxD, if any. This means that the receiver will still be synchronized with the start and stop bits. For example, if the receiver is enabled via Command Register 3 in the middle of receiving a serial character, the character will still be assembled correctly. When the receiver is disabled the last character received will remain in the receive buffer. On power-up the value in the receive buffer is undefined.

Whenever a character length of fewer than 8 bits is programmed, the most significant bits of a received character will read as zero. Also, the receiver will only check the first stop bit of any character, regardless of how many stop bits are programmed into the device.

Receive Break Detect

A Receive Break occurs when RxD remains in the space state for one character time, including the parity bit (if any) and the first stop bit. The MUART will set the Break Detect status bit (BD) when it receives a break. The Break Detect status bit is set after the middle of the first stop bit. If the MUART detects a break it will inhibit the receive buffer load pulse, thus the receive buffer will not be loaded with the null character, and none of the four status bits (PE, OE, FE, and RBF) will be updated. The last character received will remain in the receive buffer. A break detect state has the same effect as disabling the receiver—they both inhibit the load pulse—therefore one can think of the break status as disabling the receiver.

The Break Detect status bit is latched. It is cleared by the rising edge of the read pulse addressed to the status register. If a break occurs, and then the RxD data line returns to the Mark state before the status register is read, the BD status bit will remain set until it is read. If RxD returns to the Mark state after the BD status bit has been read true, the BD status bit will be reset automatically without reading the status register.

The receive break detect logic of the MUART is independent of whether the receiver is enabled or disabled; therefore even if the receiver is disabled the MUART will recognize a break. When the RxD line returns to the Mark state after a break, the 8256 will be in the start bit search mode.

If the receiver interrupt level is enabled, break will generate an interrupt request regardless of whether the receiver is enabled. Another receive interrupt will not be generated until the RxD pin returns to the Mark state.

Transmitter Section of the UART

The serial asynchronous transmitter section of the MUART consists of a transmit buffer, a transmit (shift) register, and the associated control logic. There are two bits in the status register which indicate the status of the transmit buffer and transmit register: TBE (transmit buffer empty) and TRE (transmit register empty).

To transmit a character, a byte is written to the transmit buffer. The transmit buffer should only be written to when TBE = 1. When the transmit register is empty and $\overline{CTS} = 0$, the character will be automatically transferred from the transmit buffer into the transmit register. The data transfer from the transmit buffer to the transmit register takes place during the transmission of the start bit. After this transfer takes place, sometime at the beginning of the transmission of the first data bit, TBE is set to 1.

When the transmitter is idle, both TBE and TRE will be set to 1. After a character is written to the transmit buffer, TBE = 0 and TRE = 1. This state will remain for a short period of time, then the character will be transferred into the transmit register and the status bits will read TBE = 1 and TRE = 0. At this point a second character may be written to the transmit buffer after which TBE = 0 and TRE = 0. TBE will not be set to 1 again until the transmit register becomes empty and is reloaded with the byte in the transmit buffer.

The transmitter can be disabled only one way—using the $\overline{CTS}$ pin. When $\overline{CTS} = 0$ the transmitter is enabled, and when $\overline{CTS} = 1$ the transmitter is disabled. If the transmitter is idle and $\overline{CTS}$ goes from 0 to 1, disabling the transmitter, TBE and TRE will remain set to 1. Since TBE = 1 a character can be written into the transmit buffer. The character will be stored in the transmit buffer but it will not be transferred to the transmit register until $\overline{CTS}$ goes low.

If $\overline{CTS}$ goes from low to high during transmission of a character, the character in transmission will be completed and TxD will return to the Mark state. If the transmitter is full (i.e., TBE and TRE = 0), the transmit shift register will be emptied but the transmit buffer will not; therefore TBE = 0 and TRE = 1.

Transmitter Break Features

The MUART has three transmit break features: Break-In Detect, Transmit Break (TBRK), and Single Character Break (SBRK).

Break-In Detect – A Break-In condition occurs when the MUART is sending a serial message and the transmission line is forced to the space state by the receiving station. Break-In is usually used with half-duplex transmission so that the receiver can signal a break to the transmitter. Port 16 must be connected externally to the transmission line in order to detect a Break-In. If transmission voltage levels other than TTL are used, then proper buffering must be provided so that Port 16 on the MUART will receive the correct polarity and voltage levels.

When Break-In Detect is enabled, Port 16 is polled internally during the transmission of the last or only stop bit of a character. If this pin is low during transmission of the stop bit, the Break Detect status bit (BD) will be set. Break-In Detect and receive Break Detect are OR-ed to set the BD status bit. (Either one can set this bit.) The distinction can be made through the interrupt controller. If the transmit and receive interrupts are enabled, a Break-In will generate an interrupt on level 5, the transmit interrupt, while Break will generate an interrupt on level 4, the receive interrupt. If RxC and TxC are used for the serial bit rates, Break-In cannot be detected.

Transmit Break – This causes the TxD pin to be forced low for as long as the TBRK bit in Command Register 3 is set. While Transmit Break is active, data transfers from the Transmit Buffer to the Transmit register will be inhibited.

If both the Transmit Buffer and the Transmit Register are full, and a Transmit Break command is issued (command register 3, TBRK = 1), the entire character in the Transmit register is sent including the stop bits. TxD is then driven low and the character in the Transmit Buffer remains there until Transmit Break is disabled (command register 3, TBRK = 0). At this time TxD will go high for one bit time and then send the character in the Transmit Buffer.

Single Character Break – This causes TxD to be set low for one character including start bit, data bits, parity bit, and stop bits. The user can send a specific number of Break characters using this feature.

If both the Transmit Buffer and the Transmit Register are full and a Send Break command is issued (command register 3, SBRK = 1) the entire character in the Transmit Register is sent including the stop bits. TxD is driven low for one complete character time followed by a high for two bit times after which the character in the Transmit Buffer is sent.

Modification Register

The modification register is used to alter two standard functions of the receiver (start bit check, and sampling time) and to enable a special indicator flag for half-duplex operation (transmitter status). Disabling start bit check means that the receiver will not return to the start bit search mode if RxD has returned to the Mark state in the center of the start bit. It will simply proceed to assemble a character from the RxD pin regardless of whether it received a false start bit or not. The modification register also allows the user to define where within the receive data bits the MUART will sample.

Parallel I/O

The MUART contains 16 parallel I/O pins which are divided into two 8-bit ports. These two parallel I/O ports (Port 1 and Port 2) can be used for basic digital I/O such as setting a bit high or low, or for byte transfers using a two-wire handshake. Port 1 is bit programmable for input or output, so any combination of the eight bits in Port 1 can be selected as either an input or an output. Port 2 is nibble programmable, which means that all four bits in the upper or lower nibble have to be selected as either inputs or outputs. For byte transfers using the two- wire handshake, Port 2 can either input or output the byte while two bits in Port 1 are used for the handshaking signals.

All of the bits in Port 1 have alternate functions other than I/O ports. As mentioned above, when using the byte handshake mode, two bits on Port 1 are used for the handshaking signals. As a result, these two bits cannot be used for general purpose I/O. The other six bits in Port 1 also have alternate functions if they are not used as I/O ports. Table 1 lists each bit from Port 1 and its corresponding alternate function.

The bits in the Port 1 Control Register select whether the pins on Port 1 are inputs or outputs. The pins on Port 1 are selected as control pins through the other programming registers which are relevant to the control signal. Configuring a bit in Port 1 as a control function overrides its definition in the Port 1 Control Register. If the pins on Port 1 are redefined as control signals, the definition of whether the pin is an input or an output in the Port 1 Control Register remains unchanged. If the pins on Port 1 are converted back to I/O pins, they assume the state which was defined in the Port 1 Control Register.

Each parallel I/O port has a latch and drivers. When the port is in the output mode, the data written to the port is latched and driven on the pins. The data which is latched in the I/O ports remains unchanged unless the port is written to again. Reading the ports, whether the port is an input or output, gates the state at the pins onto the data bus. Writing to an input port has no effect on the pin, but the data is stored in the latch and will be output if the direction on the pin is changed later. Writing to a control pin on Port 1 has the same effect as writing to an input pin. If pins 2, 3, 5, and 6 in Port 1 are used for control signals, the contents of the respective output latches will be read, not the state of the control signals. If pins 0, 1, and 7 on

Table 1. Port 1 Control Signals

Pin Symbol	Pin Number	Control Function	Condition
P10 P11	39 38	$\overline{ACK}$ Control signals for Port 2 $\overline{OBF}$ 8-bit handshake output	Mode register P2C2 - P2C0 = 101
P10 P11	39 38	$\overline{STB}$ Control signals for Port 2 $\overline{IBF}$ 8-bit handshake input	Mode register P2C2 - P2C0 = 100
P12	37	Event counter 2 clock input	Mode register CT2 = 1
P13	36	Event counter 3 clock input	Mode register CT3 = 1
P14	35	Internal baud rate generator clock output	Mode word P2C0 - P2C2 = 111 Port 1 control word P14 = 1 Command Register 2 B3 - B0 $\geq$ 3H
P15	34	Timer 5 trigger input	Mode register T5C = 1
P16	33	Break-In detection input	Command Register 1 BRKI = 1
P17	32	External edge sensitive interrupt input	Command Register 1 BITI = 1

Port 1 are used for control signals, the state of the control signals will be read. If pin 4 on Port 1 is used as a test output for the internal baud rate, this clock signal will be output through the output latch, thus the information in the output latch will be lost.

The Two-Wire Byte Handshake

The 8256 can be programmed, via the Mode Register, to implement an input or output two-wire byte handshake. When the Mode Register is programmed for the byte handshake, Port 2 is used to transmit or receive the byte, and pins P10 and P11 are used for the two handshake control signals. Figures 4 and 5 on pages 7 through 10 show a block diagram and timing signals for the two-wire handshake input and output.

To set up the two-wire handshake output using interrupts one must first program the Mode Register, and then enable the interrupt via the interrupt mask register. An interrupt will not occur immediately after the two-wire handshake interrupt is enabled. The interrupt is triggered by the rising edge of $\overline{ACK}$. There are two ways to generate the first interrupt. Either the first data byte must be written to Port 2 and completely transferred before an interrupt will occur, or the two-wire handshake interrupt is enabled while $\overline{ACK}$ is low, and then $\overline{ACK}$ goes high.

Event Counters/Timers

The MUART's five 8-bit programmable counters/timers are binary presettable down counters. The distinction between timer and counter is determined by the clock source. A timer measures an absolute time interval, and its input clock frequency is derived from the MUART's system clock. A counter's input clock frequency is derived from a pulse applied to an external pin. The counter is decremented on the rising edge of this pulse.

When the counters/timers are configured as timers their clock source passes through two dividers: the system clock prescaler, and the timer prescaler. As mentioned before, the system clock prescaler normalizes the internal system clock to 1.024 MHz. The timer prescaler receives this normalized system clock and devides it down to either 1 kHz or 16 kHz, depending

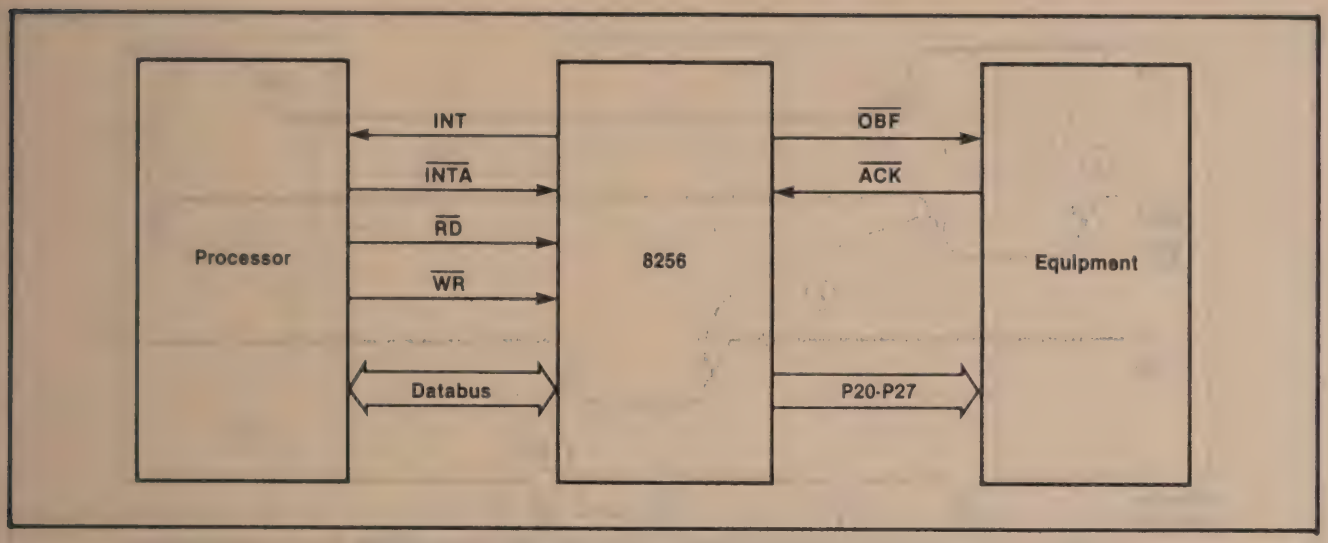


Figure 4. Block Diagram of Handshake Output

on how Command Register 1 is programmed. If more timing resolution is needed the clock frequency can be input externally through the I/O ports.

By programming the Mode Register, four of the 8-bit counters/timers can be cascaded to form two 16-bit counters. Counters/timers 3 and 5 can be cascaded together, and counters/timers 2 and 4 can be cascaded together. Counters/timers 2 and 3 are the lower bytes, while counters/timers 4 and 5 are the upper bytes in the cascaded mode.

Each counter can be loaded with an arbitrary initial value. Timer 5 is the only timer which has a special save register which holds its initial value. Whenever Timer 5 is loaded with an initial value the special save register is also loaded with this value. Timer 5 can be reloaded to its initial value from the detection of a high-to-low transition on Port P15.

The counters are decremented on the first rising edge of the clock after the initial value has been loaded. The setup time for loading the counter when using an external clock is specified in the data sheet. When using internal clocks, the user has no way of knowing the phase relationship of the clock to the write pulse; therefore the timing accuracy is one clock period.

The timers are counting continuously, and an interrupt request is issued any time a single counter or pair of cascaded counters reaches zero. If the timers are going to be used with interrupts, then the programmer should first load the timer with the initial value, then enable the interrupt. If the programmer enables the interrupt first, it is possible that the interrupt will occur before the initial value is loaded. When an interrupt from any one of the timers occurs, the corresponding bit in the interrupt mask register is automatically reset, preventing further interrupt requests from occuring.

The event counters/timers can be used in the following modes of operation:

Timer 1
— Serves as an 8-bit timer.

Event Counter/Timer 2
— Serves as an 8-bit timer or event counter, or cascaded with Timer 4 as a 16-bit timer or event counter.

Event Counter/Timer 3
— Serves as an 8-bit timer or event counter, or cascaded with Timer 5 as a 16-bit timer or event counter, with the additional modes of operation selectable for Timer 5.

Timer 4
— Serves as an 8-bit timer, or cascaded with Event Counter/Timer 2 as a 16-bit timer or event counter.

Timer 5
1) Non-retriggerable 8-bit timer
2) Retriggerable 8-bit timer whose initial value is loaded from a save register which starts following the negative transition of an external signal. Subsequent transitions of this signal after the counting has started, reloads the initial value and restarts the counting.
3) Cascaded with Event Counter/Timer 3, non-retriggerable 16-bit timer, which can be loaded with an initial value by two write operations.

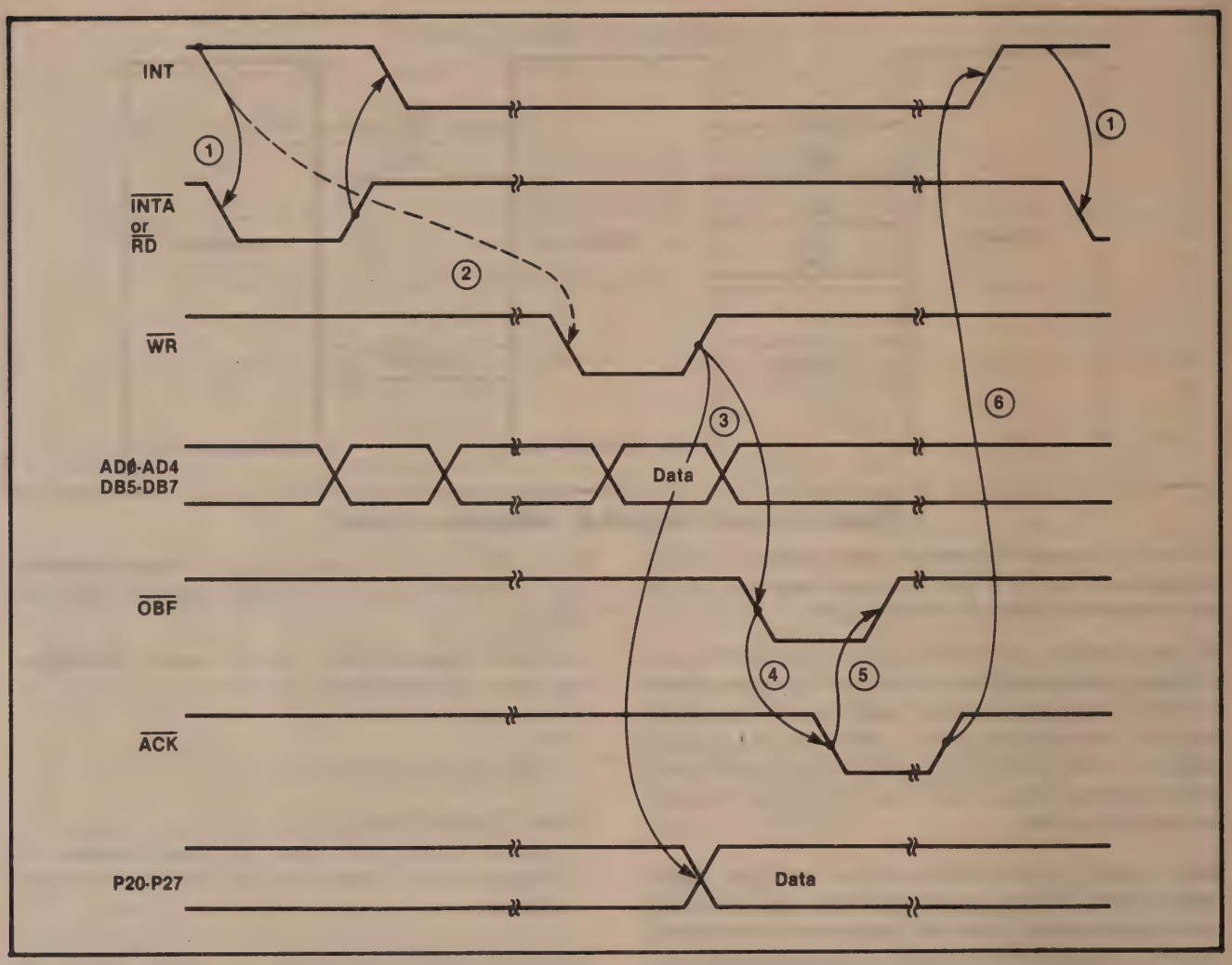


Figure 4a. Timing of Handshake Output

1. The 8256 signals with INT that the equipment has accepted the last character and that the output latches are empty again.
2. Thereupon, the microprocessor transfers the next data to the 8256.
3. The rising edge of WR latches the data into port 2 (P20...P27) and "Output Buffer Full" (OBF) is set which indicates that a new byte is available.
4. The equipment acknowledges with the falling edge of $\overline{ACK}$ that it recognized $\overline{OBF}$.
5. Thereupon, the 8256 releases $\overline{OBF}$.
6. The equipment acknowledges the data transfer with a rising edge of $\overline{ACK}$ which causes the 8256 to set INT.

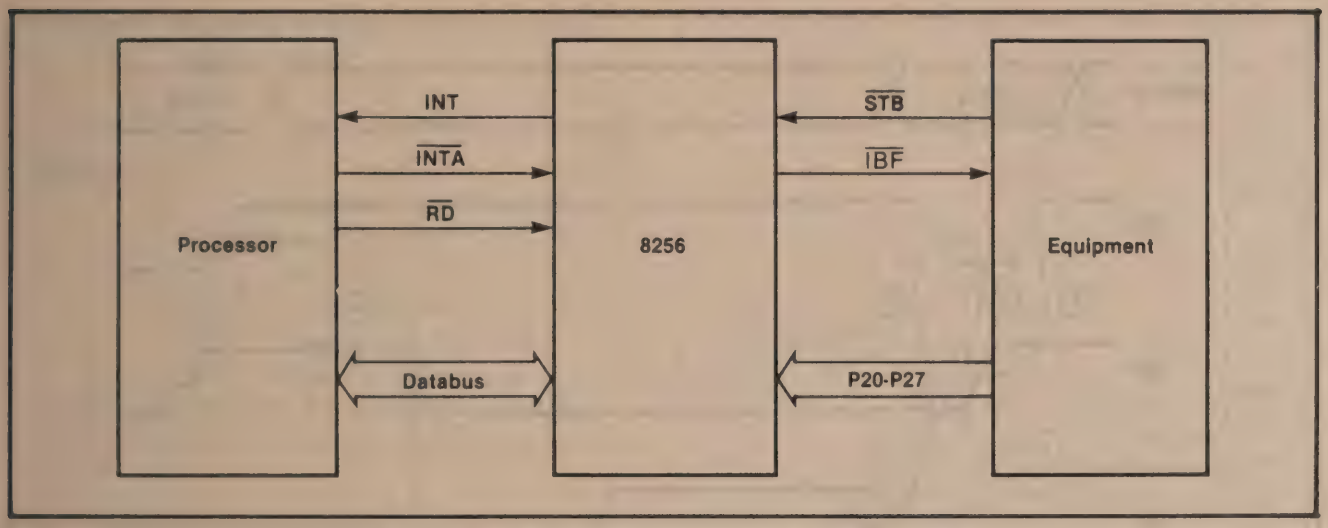


Figure 5. Block Diagram of Handshake Input

4) Cascaded with event counter/timer 3, non-retriggerable 16-bit event counter, which can be loaded with an initial value by two write operations.
5) Cascaded with Event Counter/Timer 3, retriggerable 16-bit timer. The most significant byte (Timer 5) will be loaded with its initial value from the save register, while the least significant byte (Event Counter/Timer 3) will be set to 0FFH automatically, Loading, starting, and retriggering operations follow the same pattern as in 2).
6) Cascaded with Event Counter/Timer 3, retriggerable 16-bit event counter. The most significant byte (Timer 5) will be loaded with its initial value from the save register, while the least significant byte (Event Counter/Timer 3) will be set to 0FFH automatically. Loading, starting, and retriggering operations follow the same pattern as in 2).

Interrupt Controller

In a microcomputer system there are several ways for the CPU to recognize that a peripheral device needs service. Two of the most common ways are the polling method and the interrupt service method.

In the polling method the CPU reads the status of each peripheral to determine whether it needs service. If the peripheral does not need service, the time the CPU spends polling is wasted; therefore this overhead results in increasing the execution time. Some systems must meet a specific request to response time such as a real time signal. In this case the programmer must guarantee that the peripheral is polled at a certain frequency. This polling frequency cannot always easily be met when the CPU must execute a main program as well as subroutines. Usually each peripheral has its own request to response time requirements; therefore the user must establish a priority scheme.

The interrupt method provides certain advantages over the polling method. When a peripheral device needs service it signals the CPU through hardware asynchronously, thus reducing the overhead of polling a device which does not need service. The CPU would typically finish the instruction it is executing, save the important registers, and acknowledge the peripheral's interrupt request. During the acknowledgment, the CPU reads a vector which directs the CPU to the starting location of the appropriate interrupt service routine. If several interrupt requests occur at the same time, special logic can prioritize the requests so that when the CPU acknowledges the interrupt, the highest priority request is vectored to the CPU.

An interrupt driven system requires additional hardware to control the interrupt request signal, priority, and vectoring. The 8256 integrates this additional hardware onto the chip. The interrupt controller on the MUART is directly compatible with the MCS-85, iAPX-86, iAPX-88, iAPX-186, iAPX-188 family of microcomputer systems, and it can also be used with other microprocessors as well. It contains eight priority levels, however, there are a total of 12 interruptable sources: 10 internal and 2 external. Since there are eight priority levels, only eight interrupts can be used at one time. The assignment of the interrupts used is selected by Command Register 1 and by the mode register. The MUART's interrupt sources have a fixed priority. Table 2 displays how the 12 interrupt sources are mapped into the 8 priority levels.

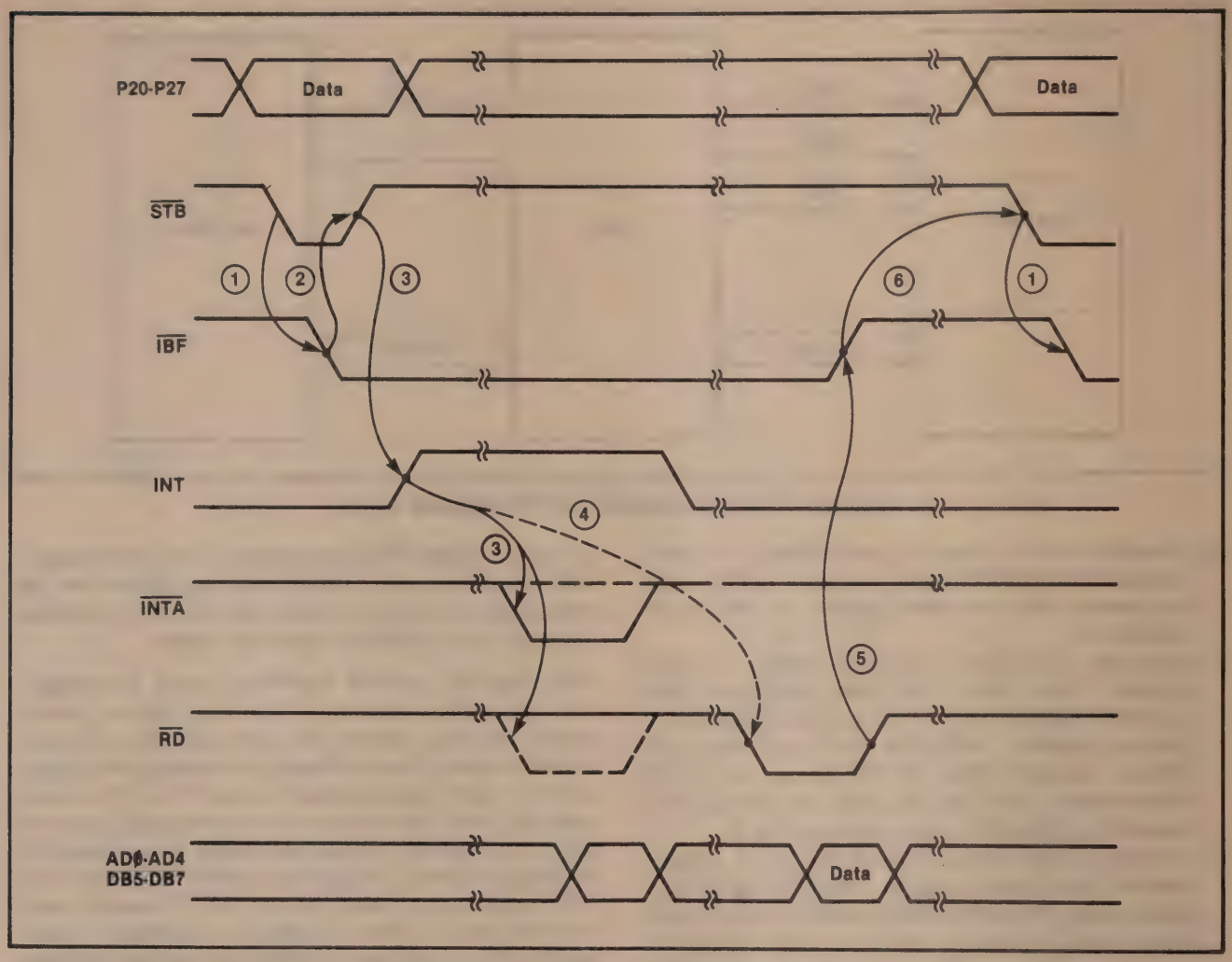


Figure 5a. Timing for Handshake Input

1. The equipment indicates with the falling edge of $\overline{STB}$ (Strobe) that a new character is available at port 2. The 8256 acknowledges the indication by activating $\overline{IBF}$ (Input Buffer Full).
2. Thereupon, the equipment releases $\overline{STB}$ and the 8256 latches the character.
3. The 8256 informs the microprocessor through INT that a new character is ready for transfer.
4. The microprocessor reads the character.
5. The rising edge of signal $\overline{RD}$ resets signal $\overline{IBF}$.
6. This action signals to the equipment that the input latches of the 8256 are empty and the next character can be transferred.

Table 2. Mapping of Interrupt Sources to Priority Levels

Priority		Source
Highest	L0	Timer 1
	L1	Timer 2 or Port Interrupt
	L2	External Interrupt (EXTINT)
	L3	Timer 3 or Timers 3 & 5
	L4	Receiver Interrupt
	L5	Transmitter Interrupt
	L6	Timer 4 or Timers 2 & 4
Lowest	L7	Timer 5 or Port 2 Handshaking

MCS®-85/8256 Interrupt Operation

The 8256 is compatible with the 8085 interrupt vectoring method when the 8086 bit in Command Register 1 of the MUART is set to 0. This is the default condition after a hardware reset. The 8085 has five hardware interrupt pins: INTR, RST 7.5, RST 6.5, RST 5.5, and TRAP. When the MUART's interrupt acknowledge feature is enabled (IAE bit 5 Command Register 3 = 1) the MUART's INT Pin 15 should be tied to the 8085's INTR, and both the 8085 and the MUART's INTA pins should be tied together. All of the interrupt pins on the 8085 except INTR automatically vector the program counter to a specified location in memory. When the INTR pin becomes active (HIGH), assuming the 8085 has interrupts enabled, the 8085 fetches the next instruction from the data bus where it has been placed by the 8256 or some other interrupt controller. This instruction is usually a Call or an RST0 through RST7. Figure 6 shows the memory locations where the 8085 will vector to based on which type of interrupt occurred.

The 8085 can receive an interrupt request any time, since its INTR input is asynchronous. The 8085, however, doesn't always acknowledge an interrupt request immediately. It can accept or disregard requests under software control using the EI (Enable Interrupt) or DI (Disable Interrupt) instructions.

At the end of each instruction cycle, the 8085 examines the state of its INTR pin. If an interrupt request is present and interrupts are enabled, the 8085 enters an interrupt machine cycle. During the interrupt machine cycle the 8085 automatically disables further interrupts until the EI instruction is executed. Unlike normal machine cycles, the interrupt machine

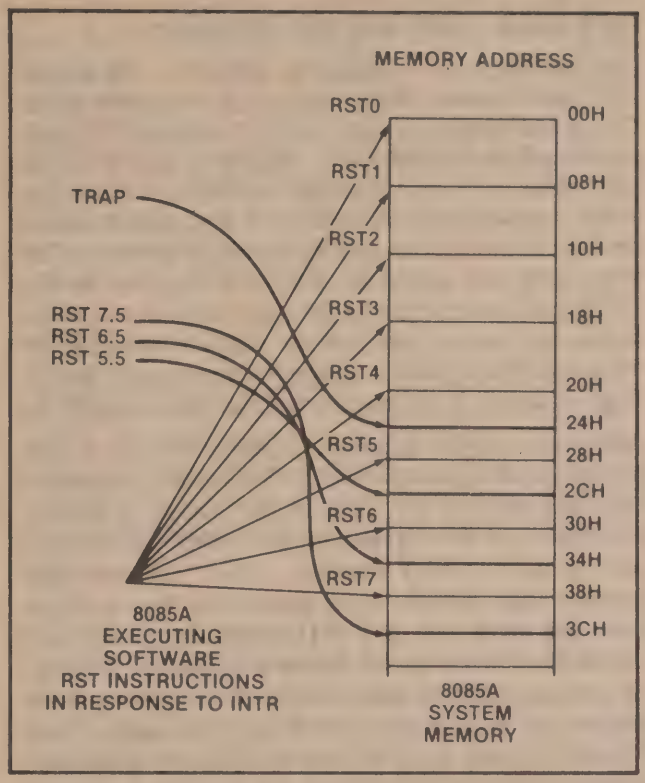


Figure 6. 8085A Hardware and Software RST Branch Locations

cycle doesn't increment the program counter. This ensures that the 8085 can return to the pre-interrupt program location after the interrupt service is completed. The 8085 issues an $\overline{\text{INTA}}$ pulse indicating that it is honoring the request and is ready to process the interrupt.

The 8256 can now vector program execution to the corresponding service routine. This is done during the first and only $\overline{\text{INTA}}$ pulse. Upon receiving the $\overline{\text{INTA}}$ pulse, the 8256 places the opcode RSTn on the data bus; where n equals 0 through 7 based on the level of the interrupt requested. The RSTn instruction causes the contents of the program counter to be pushed onto the stack, then transfers control to the instruction whose address is eight times n, as shown in Figure 6.

Note that because interrupts are disabled during the interrupt acknowledge sequence, the EI instruction must be executed in either the service routine or the main program before further interrupts can be processed.

For additional information on the 8085 interrupt operation and the RSTn instruction, refer to the *MCS-85 User's Manual.*

iAPX-86/88 - 8256 Interrupt Operation

The MUART is compatible with the 8086/8088 method of interrupt vectoring when the 8086 bit in Command Register 1 is set to 1. The MUART's INT pin is tied to the 8086/8088 INTR pin, and its $\overline{\text{INTA}}$ pin connected to the 8086/88's $\overline{\text{INTA}}$ pin. Like the 8085, the 8086/8088's INTR pin is also asynchronous so that an interrupt request can occur at any time. The 8086/8088 can accept or disregard requests on the INTR pin under software control instructions. These instructions set or clear the interrupt-enabled flag IF. When the 8086/8088 is powered-on or reset, the IF flag is cleared, disabling external interrupts on INTR.

Although there are some basic similarities, the actual processing of interrupts with an 8086/8088 is different from the 8085. When an interrupt request is present and interrupts are enabled, the 8086/8088 enters its interrupt acknowledge machine cycle. The interrupt acknowledge machine cycle pushes the flag registers onto the stack (as in PUSHF instruction). It then clears the IF flag, which disables interrupts. Finally, the contents of both the code segment register and the instruction pointer are pushed onto the stack. Thus, the stack retains the pre-interrupt flag status and program location which are used to return from the service routine. The 8086/8088 then issues the first of two INTA pulses which signals the 8256 that the 8086/8088 has honored its interrupt request.

The 8256 is now ready to vector program execution to the appropriate service routine. Unlike the 8085 where the first INTA pulse is used to place an instruction on the data bus, the first INTA pulse from the 8086/8088 is used only to signal the 8256 of the honored request. The second INTA pulse causes the 8256 to place a single interrupt vector byte onto the data bus. The 8256 places the interrupt vector bytes 40H through 47H corresponding to the level of the interrupt to be serviced. Not used as a direct address, this interrupt vector byte pertains to one of 256 interrupt "types" supported by the 8086/8088 memory. Program execution is vectored to the corresponding service routine by the contents of a specified interrupt type.

All 256 interrupt types are located in absolute memory locations 0 through 3FFH which make up the 8086/8088's interrupt vector table. Each type in the interrupt vector table requires 4 bytes of memory and stores a code segment address and an instruction pointer address. Figure 7 shows a block diagram of the interrupt vector table. When the 8086/8088 receives an interrupt vector byte, it multiplies its value by four to acquire the address of the interrupt type.

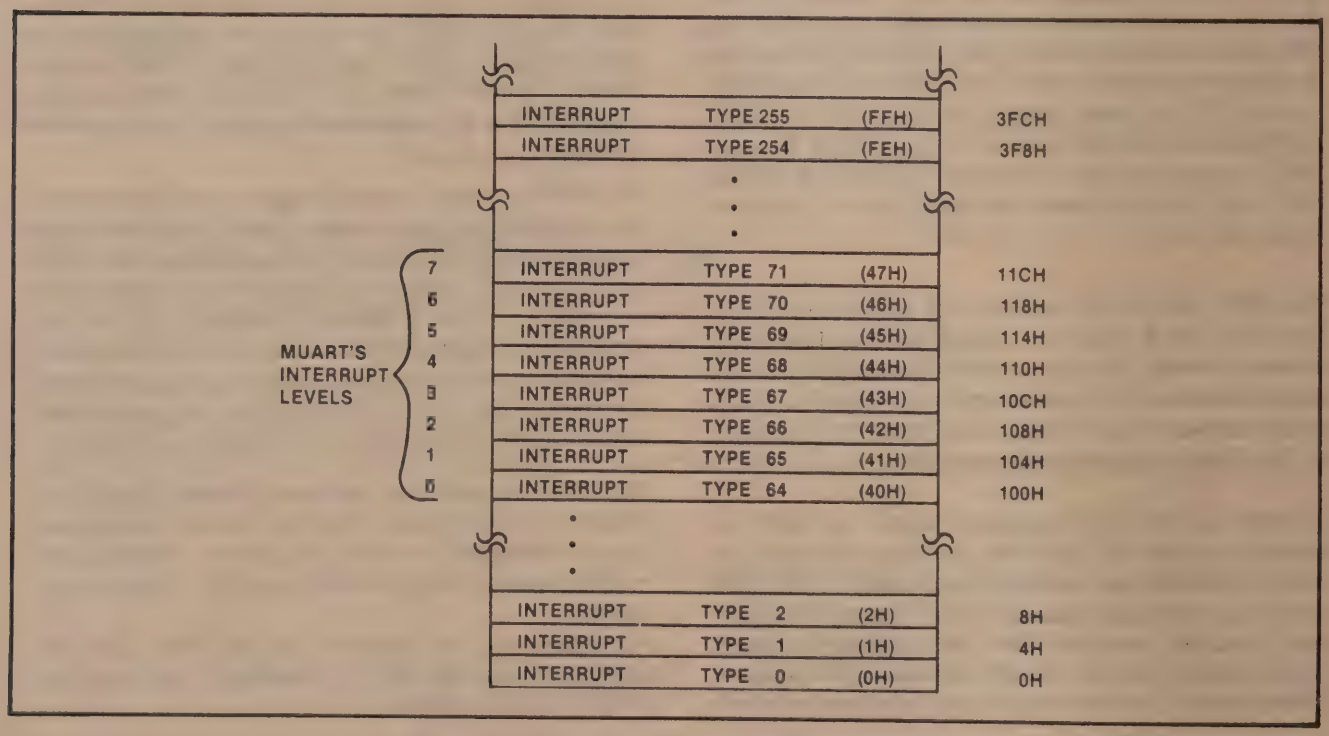


Figure 7. 8086/8088 Interrupt Vector Table

Once the service routine is completed the main program may be reentered by using an IRET (Interrupt Return) instruction. The IRET instruction will pop the pre-interrupt instruction pointer, code segment and flags off the stack. Thus the main program will resume where it was interrupted with the same flag status regardless of changes in the service routine. Note especially that this includes the state of the IF flag; thus interrupts are re-enabled automatically when returning from the service routine. For further information refer to the *iAPX 86,88 User's Manual.*

Using the 8256's Interrupt Controller Without $\overline{\text{INTA}}$

There are several configurations where the 8256 will not have an $\overline{\text{INTA}}$ signal connected to it. Some examples are when using the 8256 with an 8051 or 8048, or when connecting the INT pin on the 8256 to the 8085's RST 7.5, RST 6.5, or RST 5.5 inputs. In these configurations the $\overline{\text{IAE}}$ bit in Command Register 3 is set to 0, and the $\overline{\text{INTA}}$ pin on the 8256 is tied high. When the interrupt occurs the CPU should branch to a service routine which reads the interrupt address register to determine which interrupt request level occured. The interrupt address register contains the level of the interrupt multiplied by four. Reading the interrupt address register is equivalent in effect to the $\overline{\text{INTA}}$ signal; it clears the INT pin and indicates to the MUART that the interrupt request has been acknowledged. After the CPU reads the value in the interrupt address register, it can add an offset to this value and branch to an interrupt vector table which contains jump instructions to the appropriate interrupt service routines. An 8085 program which demonstrates this routine is given is Figure 8.

Table 3 summarizes the priority levels and the interrupt vectors which the 8256 sends back to the CPU. Note that when using Timer 1 there is a conflict present between RST0 in the 8085 mode and a hardware reset, because both expect instructions starting at address 0H. However, there is a way to distinguish between the two. After a hardware reset, all control registers are reset to a value of 0H; therefore when using Timer 1, Reset and RST0 can be distinguished by reading one of the control registers of the 8256 which has not been programmed with a value of 0H. The control registers will contain the previously programmed values if RST0 occurs.

Interrupt Registers

The 8256's interrupt controller has several registers associated with it: an Interrupt Mask Register, an Interrupt Address Register, an Interrupt Request Register, an Interrupt Service Register, and a Priority Controller. Only the Interrupt Mask Registers and the Interrupt Address Register can be accessed by the user.

Interrupt Mask Registers

The Interrupt Mask Registers consist of two write registers — the Set Interrupts Register and Reset Interrupts Register, and one read register — the Interrupt Enable Register. Each one of the eight levels of interrupts may be individually enabled or disabled through these registers. Writing a one to any of the bits in the Set Interrupts Register enables the corresponding interrupt level, while writing a one to a bit in the Reset Interrupts Register disables the corresponding interrupt level. Reading the Interrupt Enable Register allows the user to determine which interrupt levels are enabled. The bits which are set to one in the Interrupt Enable Register correspond to the levels which are enabled. All of the interrupt levels will remain enabled until disabled by the Reset Interrupts Register except the counter/timer interrupts which automatically disable themselves when they reach zero.

```
INTA:  IN     INTADD      ;Read the Interrupt Address Register
       MOV    L, A        ;Put the interrupt address in HL
       XRA    A
       MOV    H, A

       LXI    B, TABLE    ;Load BE with the interrupt table offset
       DAD    B           ;Add the offset to the interrupt address
       PCHL               ;Jump to the interrupt vecor table
```

Figure 8. Software Interrupt Acknowledge Routine

Table 3. Assignment of Interrupt Levels to Interrupt Sources

Interrupt Level	Restart Command 8085 mode	Interrupt Vector 8086 mode	Interrupt Address	Trigger Mode	Sources (Only one source can be assigned at any time)	Selection by
Highest Priority 0	RST0	40H	0H	edge	Timer 1	–
1	RST1	41H	4H	edge	Event Counter/Timer 2 or external interrupt request on Port 1 P17	Command word 1 BITI (bit 2)
2	RST2	42H	8H	level	Input EXTINT	–
3	RST3	43H	CH	edge	Event Counter/Timer 3 or cascaded event counters/ timers 3 and 5	Mode word T35 (bit 7)
4	RST4	44H	10H	edge	Serial receiver	–
5	RST5	45H	14H	edge	Serial transmitter	–
6	RST6	46H	28H	edge	Timer 4 or cascaded event counters/timers 2 and 4	Mode word T24 (bit 6)
7 Lowest Priority	RST7	47H	1CH	edge	Timer 5 or Port 2 with handshaking interrupt request	Mode word P2C2 – P2C0 (bits 2...0)

Note:
If no interrupt requests are pending and $\overline{\text{INTA}}$ cycle occurs, interrupt level 2 will be the default value vectored to the CPU.

Interrupt requests occurring when the corresponding interrupt level is disabled are lost. An interrupt will only occur if the interrupt is enabled before the interrupt request occurs.

Interrupt Address Register

The Interrupt Address Register contains an identifier for the currently requested interrupt level. The numerical value in this register is equal to the interrupt level mutliplied by four. It can be used in lieu of an $\overline{\text{INTA}}$ signal to vector the CPU to the appropriate interrupt service routine. Reading this register has the same effect as the $\overline{\text{INTA}}$ pulse: it clears the INT pin and indicates an interrupt acknowledgement to the MUART. If the Interrupt Address Register is read while no interrupts are pending, the external interrupt EXTINT will be the default value, 08H.

Interrupt Request Register

The Interrupt Request Register latches all pending interrupt requests unless they are masked off. The request is set whenever the associated event occurs.

Interrupt Service Register

In the fully nested mode of operation, every interrupt request which is granted service is entered into this register. The appropriate bit will be set whenever the interrupt is acknowledged by $\overline{\text{INTA}}$ or by reading the Interrupt Address Register. At the same time, the corresponding bit in the Interrupt Request Register is reset. The Interrupt Service Register bit remains set until the microcomputer transfers the End Of Interrupt command (EOI) to the device by writing it into Command Register 3. In the normal mode the bits in the Interrupt Service Register are never set.

Priority Controller

The priority controller selects the highest priority request in the Interrupt Request Register from up to eight requests pending. If the $\overline{\text{INTA}}$ signal is enabled and becomes active, the priority controller will cause the highest priority level in the Interrupt Request Register to be vectored back to the CPU, regardless of whether the 8256 is in the normal mode or the nested mode. In the normal mode, if any bits are set in the Interrupt Request Register, the INT pin is activated. The highest priority level in the Interrupt Request register will be transferred to the Interrupt Address Register at the same time the interrupt request occurs. In the Fully Nested mode, the priorities of all pending requests are compared to the priorities in the Interrupt Service Register. If there is a higher priority in the Interrupt Request Register than in the Interrupt Service Register, the INT signal will be activated and the new interrupt level will be loaded into the Interrupt Address Register.

Interrupt Modes

There are two modes of operation for the interrupt controller: a normal mode and a fully nested mode. In the normal mode the CPU should only be a maximum of one interrupt level deep; therefore, the CPU can be interrupted only while in the main program and not while in an interrupt service routine. In the fully nested mode it is possible for the CPU to be nested up to eight interrupt levels deep. Using the fully nested mode, the MUART will activate the INT pin only when a higher priority than the one in service is requested. The fully nested mode is used to protect high priority interrupt service routines from being interrupted by equal or lower priority requests.

Normal Mode

In the normal mode of operation the 8256 will activate the INT pin whenever any of the bits in the Interrupt Request Register are set. The bits in the Interrupt Request Register can be set only if the corresponding interrupts are enabled. If more than one interrupt request bit is set, the MUART will always place the highest priority level in the Interrupt Address Register and vector this level to the CPU during an $\overline{\text{INTA}}$ cycle. When the CPU acknowledges the interrupt request, using either the $\overline{\text{INTA}}$ signal or by reading the Interrupt Address Register, the corresponding Interrupt Request Register bit is reset. Since the Interrupt Service Register bits are never set, there is no indication in the MUART that an interrupt service routine is in progress. Therefore, the priority controller will interrupt the CPU again if any of the interrupt request bits are set, regardless of whether the next request is a higher, lower, or equal priority.

The implied way to design a program using the normal mode is to have the CPU's interrupt flag enabled during portions of the main program, but to leave the interrupt flag disabled while the CPU is executing code in an interrupt service routine. This way, the CPU can never be interrupted in an interrupt service routine. Upon completion of an interrupt service routine the program can enable the CPU's interrupt flag, then return to the main program.

Figure 9 shows an example of how the normal mode of interrupts may operate. As the CPU begins executing code in the main program, certain I/O ports, variables, and arrays need to be initialized. During this time the CPU's interrupt flag is disabled. Once the program has completed the initialization routine and can accept an interrupt, the interrupt flag is enabled. In the 8085 this is done with the assembly language instruction EI, and on the 8086 with STI.

A short time later, an interrupt request comes in on Level 4. Since the CPU's interrupt flag is enabled, the interrupt acknowledge signal is activated and the CPU branches off to Interrupt Service Routine 4. While the CPU is executing code in Interrupt Service Routine 4, an interrupt request comes in on Level 6 and then a short time later on Level 2. The 8256 activates the INT signal; however, the CPU ignores this because its interrupt flag is disabled. Upon returning to the main program the interrupt flag is enabled. When the interrupt acknowledge signal is activated, the MUART places the highest priority interrupt request on the data bus regardless of the order in which the requests came in. Therefore, during the interrupt acknowledge the MUART vectors the indirect address for Interrupt Level 2. The INT signal is not cleared after the acknowledge because there is still a pending interrupt.

The normal mode of operation is advantageous in that it simplifies programming and lowers code requirements within interrupt routines; however, there are also several disadvantages. One disadvantage is that the interrupt response time for higher priority interrupts may be excessive. For example, if the CPU is executing code in an interrupt service routine during a higher priority request, the CPU will not branch off to the higher priority service routine until the current interrupt service routine is completed. This delay time may not be acceptable for interrupts such as the serial receiver or a real time signal. For these cases the MUART provides the nested mode.

Nested Mode

In the nested mode of operation, whenever a bit in the Interrupt Request Register is set, the Priority Con-

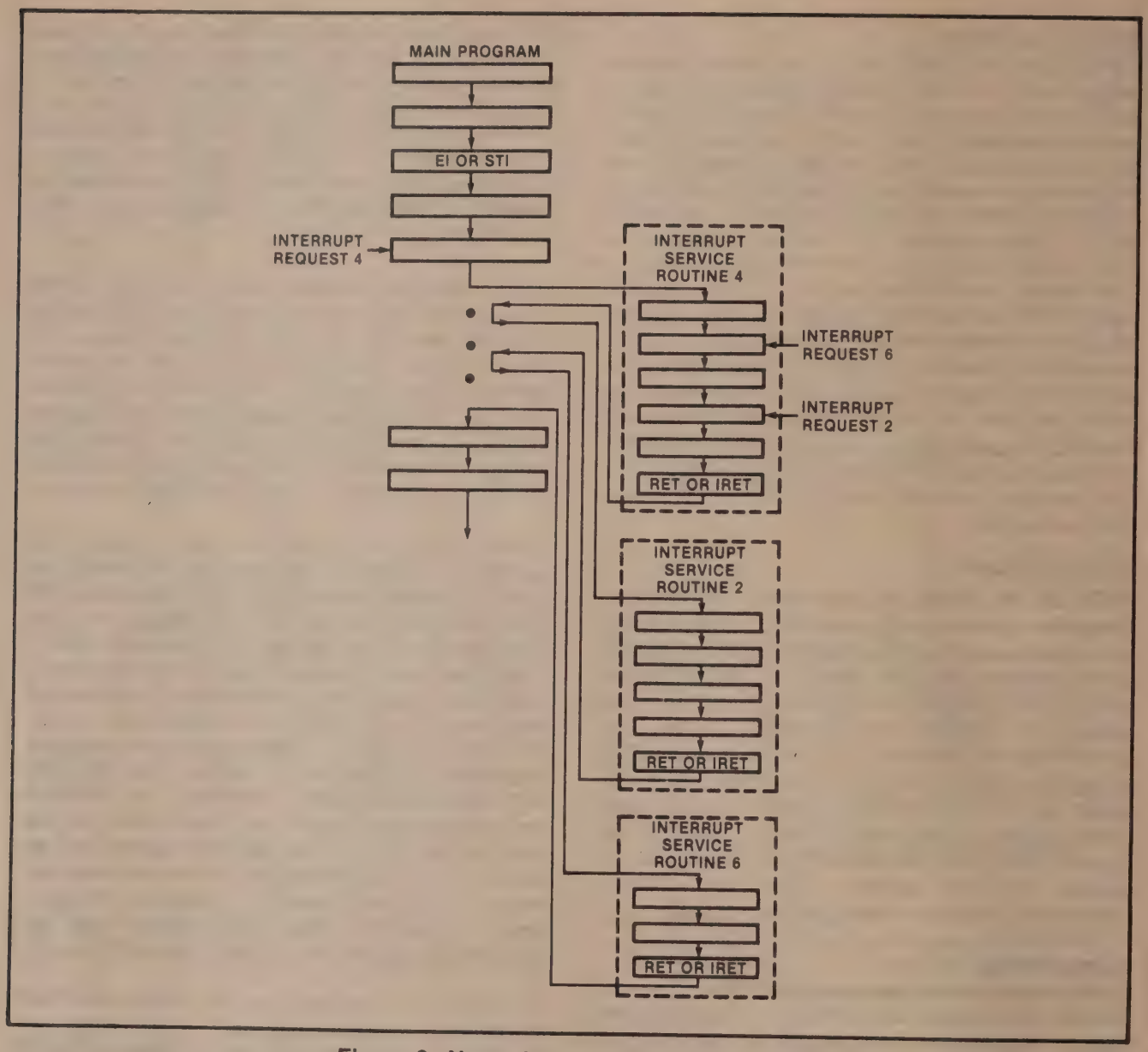


Figure 9. Normal Interrupt Mode Example

troller compares the Interrupt Request Register to the Interrupt Service Register. If the bit set in the Request Register is of a higher priority than the highest priority bit set in the Service Register, the MUART will activate the INT signal and update the Interrupt Address Register. If the bit in the Request Register is of equal or lower priority than the highest priority bit set in the Service Register, the INT signal will not be activated. When an $\overline{\text{INTA}}$ signal is activated or the Interrupt Address Register is read, the corresponding bit in the Request Register which caused the INT signal to be asserted is reset and set in the Service Register. When an EOI (End Of Interrupt) command is issued, the highest priority bit in the Service Register is reset.

Figure 10 shows an example of the program flow using the nested mode of interrupts. During the main program an interrupt request is generated from Level 4. Since the interrupt flag is enabled, the interrupt acknowledge signal is activated, and the microprocessor is vectored to Service Routine 4. During Service Routine 4, Level 2 requests an interrupt. Since Level 2 is a higher priority than Level 4, the 8256 activates its INT signal. An interrupt

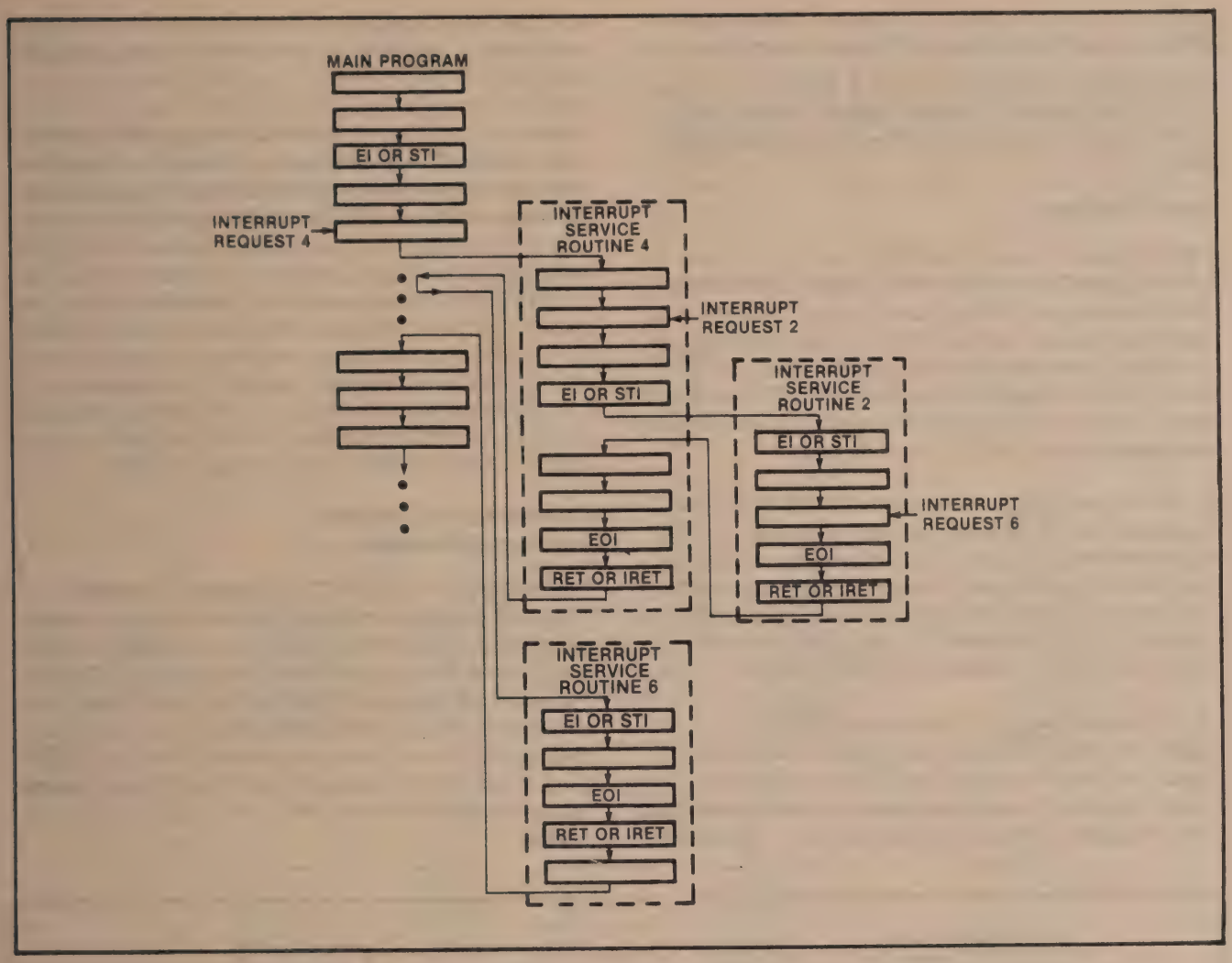


Figure 10. Fully Nested Interrupt Mode Example

acknowledge is not generated because the interrupt flag is disabled. This section of code in Service Routine 4 is protected and cannot be interrupted. A protected section of code may reinitialize a timer, take a sample, or update a global variable. When the interrupt flag is enabled the microprocessor acknowledges the interrupt and vectors into Service Routine 2. Service Routine 2 immediately enables the interrupt flag because it does not have a protected section of code. During Service Routine 2, Interrupt Request 6 is generated. However, the MUART will not interrupt the microprocessor until service routines 2 and 4 have issued the EOI command.

Edge Triggering

The MUART has a maximum of two external interrupts—EXTINT and P17. EXTINT is a dedicated interrupt pin which is level triggered, where P17 is either an I/O port or an edge triggered interrupt. If P17 is selected as an interrupt through Command Register 1 and its interrupt level is enabled, it will generate an interrupt when the level on this pin changes from low to high. The edge triggered mode incorporates an edge lockout feature. This means that after the rising edge of an interrupt request and the acknowledgment of the request, the positive level on

P17 won't generate further interrupts. Before another interrupt can be generated P17 must return low.

External devices which generate a pulse for an interrupt request can use the edge triggered mode as long as the minimum high time specified in the data sheet is met.

Level Triggering

The external interrupt (EXTINT pin 16) is the only level triggered interrupt on the MUART. The 8256 will recognize any active (high) level on the EXTINT as an interrupt request. The EXTINT pin must stay high until a short time after the rising edge of the first $\overline{\text{INTA}}$ pulse. If the voltage level on the EXTINT pin is high then goes low, the bit in the interrupt request register corresponding to EXTINT will be reset.

In the normal mode of operation if EXTINT is still high after the $\overline{\text{INTA}}$ pulse has been activated, the INT signal will remain active. If the microprocessor's interrupt flag is immediately reenabled, another interrupt will occur. Unless repeated interrupt generation is desired, the programmer should not reenable the CPU's interrupt flag until EXTINT has gone low.

In the nested mode of operation, if EXTINT is still high after the $\overline{\text{INTA}}$ pulse has been activated, the INT signal will not be reactivated. This is because in the nested mode only a higher priority interrupt than the one being serviced can activate the INT signal. The EXTINT pin should go inactive (low) before the EOI command is issued if an immediate interrupt is not desired.

Depending upon the particular design and application, the EXTINT pin has a number of uses. For example, it can provide repeated interrupt generation in the normal mode. This is useful in cases when a service routine needs to be continually executed until the interrupt request goes inactive. Another use of the EXTINT pin is that a number of external interrupt requests can be wire-ORed. This can't be done using P17, for if a device makes an interrupt request while P17 is high (from another request), its transition will be shadowed. Note that when a wire-OR'ed scheme is used, the actual requesting device has to be determined by the software in the service routine.

Cascading the MUART's Interrupt Controller

Cascading the MUART's interrupt controller is necessary in an interrupt driven system which contains more than one interrupt controller, such as a system using more than one MUART, or using a MUART with another interrupt controller like the 8259A. For a system which uses several MUART's, one of them is tied directly to the microprocessor's INT and $\overline{\text{INTA}}$ pins, while the remaining MUARTs are daisy-chained using the EXTINT and INT pins. This is shown in Figure 11.

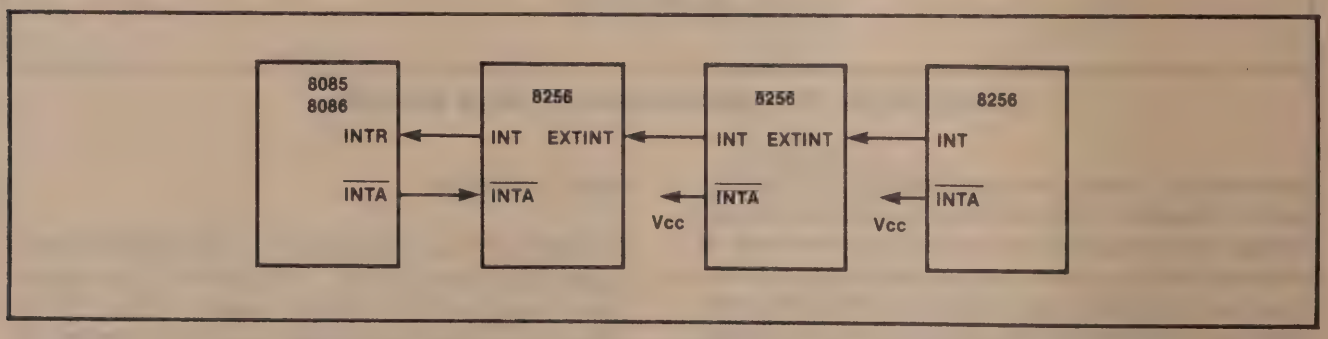


Figure 11. Cascading the MUART's Interrupt Controller

Using the configuration in Figure 11, when the microprocessor receives an interrupt, it generates an interrupt acknowledge and branches into an interrupt service routine. For the interrupt service routine of the external interrupt, EXTINT Level 2, the microprocessor will read the next MUART's interrupt address register and branch to the appropriate service routine. In effect, this would be a software interrupt acknowledge. An example of this type of interrupt acknowledge is given in Figure 8. If the last MUART in the chain indicated an external interrupt, the microprocessor would simply return to the main program; however, this would be an error condition caused by a spurious interrupt. A flow chart of the software to handle cascaded interrupts is given in Figure 12.

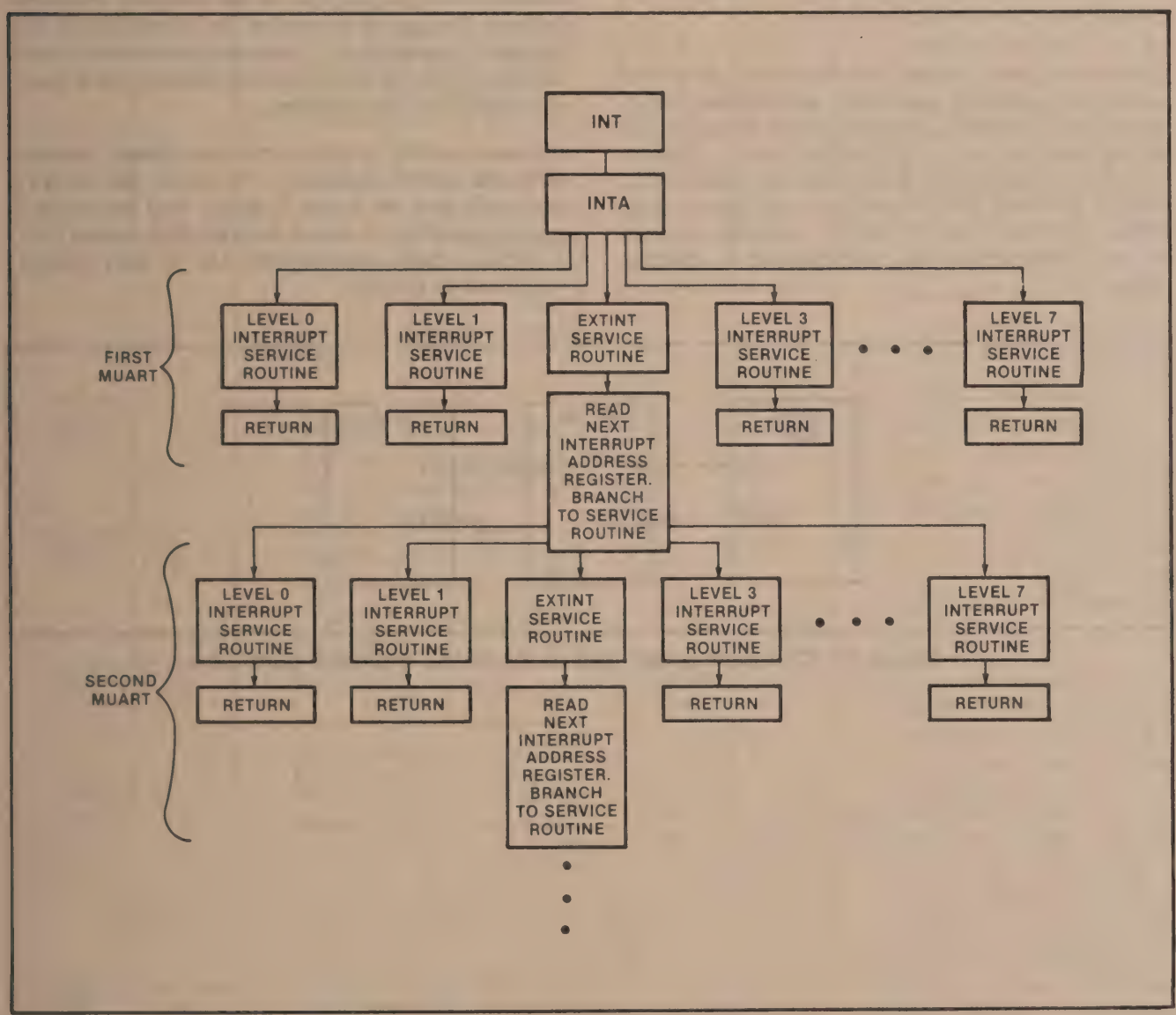


Figure 12. Flow Chart to Resolve Interrupt Request When Cascading MUART Interrupt Controllers

Some consideration should be given to the priority of the interrupts when cascading MUARTs. If all of the MUART's Level 0 and Level 1 interrupts are disabled, the highest priority interrupt is the EXTINT. In this case the last MUART in the chain would have the highest priority; however, it would take the longest time to propagate back to the CPU. If, however, Level 0 or Level 1 interrupts were enabled, the closer to the microprocessor the MUART is, the higher the priority these two levels would have.

When using the 8256 interrupt controller along with some other interrupt controller, such as the 8259A, the MUART's INT signal would simply be tied to one of the interrupt controller's request inputs. The service routine for the MUART's interrupt request would initially perform the software interrupt acknowledge before servicing the MUART's interrupt request. A block diagram of this configuration is given in Figure 13.

Polling the MUART

If interrupts are not used, the only other way to control the MUART is to poll it. It is still possible to use the priority structure of the MUART with polling. In this mode of operation the MUART's INT signal (Pin 15) is not used, and the $\overline{INTA}$ pin is tied high. Since the INT pin's level is duplicated in the MSB of the Status Register, a program can poll this bit. When it becomes set, the program could read the Interrupt Address Register to determine the cause. Either the normal or nested mode of operation can be used. Note that the functions used with this polled method must have their interrupts enabled.

It is also possible to poll the counters/timers, parallel I/O, and UART separately. To control the UART, one could poll the Status Register. Byte handshakes with the parallel I/O can be controlled by polling Port 1. Finally, each counter/timer has its own register which can be polled.

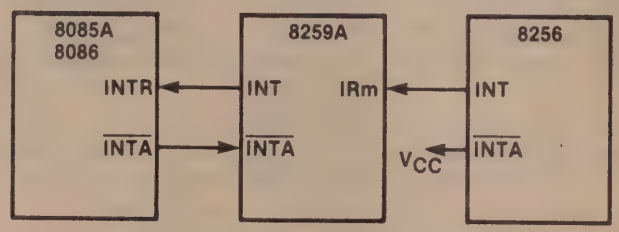


Figure 13. Connecting the 8256 to the 8259A Interrupt Controller

PIN DESCRIPTIONS

Symbol	Pin No.	Type	Name and Function
AD0-AD4 DB5-DB7	1-5 6-8	I/O	**Address/Data:** Three-state address/data lines which interface to the lower 8 bits of the microprocessor's multiplexed address/data bus. The 5-bit address is latched on the falling edge of ALE. In the 8-bit mode, AD0-AD3 are used to select the proper register, while AD1-AD4 are used in the 16-bit mode. AD4 in the 8-bit mode is ignored as an address, while AD0 in the 16-bit mode is used as a second chip select, active low.
ALE	9	I	**Address Latch Enable:** Latches the 5 address lines on AD0-AD4 and $\overline{CS}$ on the falling edge.
$\overline{RD}$	10	I	**Read Control:** When this signal is low, the selected register is gated onto the data bus.
$\overline{WR}$	11	I	**Write Control:** When this signal is low, the value on the data bus is written into the selected register.
RESET	12	I	**Reset:** An active high pulse on this pin forces the chip into its initial state. The chip remains in this state until control information is written.
$\overline{CS}$	13	I	**Chip Select:** A low on this signal enables the MUART. It is latched with the address on the falling edge of ALE, and $\overline{RD}$ and $\overline{WR}$ have no effect unless $\overline{CS}$ was latched low during the ALE cycle.

Symbol	Pin No.	Type	Name and Function
$\overline{INTA}$	14	I	**Interrupt Acknowledge:** If the MUART has been enabled to respond to interrupts, this signal informs the MUART that its interrupt request is being acknowledged by the microprocessor. During this acknowledgement the MUART puts an RSTn instruction on the data bus for the 8-bit mode or a vector for the 16-bit mode.
INT	15	O	**Interrupt Request:** A high signals the microprocessor that the MUART needs service.
EXTINT	16	I	**External Interrupt:** An external device can request interrupt service through this input. The input is level sensitive (high), therefore it must be held high until an $\overline{INTA}$ occurs or the interrupt address register is read.
CLK	17	I	**System Clock:** The reference clock for the baud rate generator and the timers.
RxC	18	I/O	**Receive Clock:** If the baud rate bits in Command Register 2 are all 0, this pin is an input which clocks serial data into the RxD pin on the rising edge of RxC. If baud rate bits in Command Register 2 are programmed from 1-0FH, this pin outputs a square wave whose rising

PIN DESCRIPTIONS (CONTINUED)

Symbol	Pin No.	Type	Name and Function
			edge indicates when the data on RxD is being sampled. This output remains high during start, stop, and parity bits.
RxD	19	I	**Receive Data:** Serial data input.
$\overline{CTS}$	21	I	**Clear To Send:** This input enables the serial transmitter. If 1, 1.5, or 2 stop bits are selected, $\overline{CTS}$ is level sensitive. As long as $\overline{CTS}$ is low, any character loaded into the transmitter buffer register will be transmitted serially. A single negative going pulse causes the transmission of a single character previously loaded into the transmitter buffer register. If a baud rate from 1-0FH is selected, $\overline{CTS}$ must be low for at least 1/32 of a bit, or it will be ignored. If the transmitter buffer is empty, this pulse will be ignored. If this pulse occurs during the transmission of a character up to the time where 1/2 of the first (or only) stop bit is sent out, it will be ignored. If it occurs afterwards, but before the end of the stop bits, the next character will be transmitted immediately following the current one. If $\overline{CTS}$ is still high when the transmitter register is sending the last stop bit, the transmitter will enter its idle state until the next high-to-low transition on $\overline{CTS}$ occurs. If 0.75 stop bits is chosen, the $\overline{CTS}$ input is edge sensitive. A negative edge on $\overline{CTS}$ results in the immediate transmission of the next character. The length of the stop bits is determined by the time interval between the beginning of the first stop bit and the next negative edge on $\overline{CTS}$. A high-to-low transition has no effect if the transmitter buffer is empty or if the time interval between the beginning of the stop bit and next negative edge is less than 0.75 bits. A high or a low level or a low-to-high transition has no effect on the transmitter for the 0.75 stop bit mode.
TxC	22	I/O	**Transmit Clock:** If the baud rate bits in command register 2 are all set to 0, this input clocks data out of the transmitter on the falling edge. If baud rate bits are programmed for 1 or 2, this input permits the user to provide a 32x or 64x clock which is used for the receiver and transmitter. If the baud rate bits are programmed for 3-0FH, the internal transmitter clock is output. As an output it delivers the transmitter clock at the selected bit rate. If 1½ or 0.75 stop bits are selected, the transmitter divider will be asynchronously reset at the beginning of each

PIN DESCRIPTIONS (CONTINUED)

Symbol	Pin No.	Type	Name and Function
			start bit, immediately causing a high-to-low transition on TxC. TxC makes a high-to-low transition at the beginning of each serial bit, and a low-to-high transition at the center of each bit.
TxD	23	O	**Transmit Data:** Serial data output.
P27-P20	24-31	I/O	**Parallel I/O Port 2:** Eight bit general purpose I/O port. Each nibble (4 bits) of this port can be either an input or an output. The outputs are latched whereas the input signals are not. Also, this port can be used as an 8-bit input or output port when using the two-wire handshake. In the handshake mode both inputs and outputs are latched.
P17-P10	32-39	I/O	**Parallel I/O Port 1:** Each pin can be programmed as an input or an output to perform general purpose I/O. All outputs are latched whereas inputs are not. Alternatively these pins can serve as control pins which extend the functional spectrum of the chip.
GND	20	PS	**Ground:** Power supply and logic ground reference.
Vcc	40	PS	**Power:** +5V power supply.

DESCRIPTION OF THE REGISTERS

The following section will provide a description of the registers and define the bits within the registers where appropriate. Table 4 lists the registers and their addresses.

Command Register 1

L1	L0	S1	S0	BRKI	BITI	8086	FRQ
	(0R)				(0W)		

FRQ — Timer Frequency Select

This bit selects between two frequencies for the five timers. If FRQ=0, the timer input frequency is 16KHz (62.5*us*). If FRQ=1, the timer input frequency is 1 KHz (1ms). The selected clock frequency is shared by all the counter/timers enabled for timing; thus, all timers must run with the same time base.

8086 — 8086 Mode Enable

This bit selects between 8085 mode and 8086/8088 mode. In 8085 mode (8086=0), A0 to A3 are used to address the internal registers, and an RSTn instruction is generated in response to the first $\overline{\text{INTA}}$. In 8086 mode (8086=1), A1 to A4 are used to address the internal registers, and A0 is used as an extra chip select (A0 must equal zero to be enabled). The response to $\overline{\text{INTA}}$ is for 8086 interrupts where the first $\overline{\text{INTA}}$ is ignored, and an interrupt vector (40H to 47H) is placed on the bus in response to the second $\overline{\text{INTA}}$.

BITI — Interrupt on Bit Change

This bit selects between one of two interrupt sources on Priority Level 1, either Counter/Timer 2 or Port 1 P17 interrupt. When this bit equals 0, Counter/Timer 2 will be mapped into Priority Level 1. If BITI equals 0 and Level 1 interrupt is enabled, a transition from 1 to 0 in Counter/Timer 2 will generate an interrupt request on Level 1. When BITI equals 1, Port 1 P17 external edge triggered interrupt source is mapped into Priority Level 1. In this case if Level 1 is enabled, a low-to-high transition on P17 generates an interrupt request on Level 1.

BRKI — Break-In Detect Enable

If this bit equals 0, Port 1 P16 is a general purpose I/O port. When BRKI equals 1, the Break-In Detect feature is enabled on Port 1 P16. A Break-In condition is present on the transmission line when it is forced to the start bit voltage level by the receiving station. Port 1 P16 must be connected externally to the transmission line in order to detect a Break-In. A

Table 4. MUART Registers

Read Registers								8085 Mode: AD3 / 8086 Mode: AD4	AD2 / AD3	AD1 / AD2	AD0 / AD1	Write Registers							
L1	L0	S1	S0	BRKI	BITI	8086	FRQ	0	0	0	0	L1	L0	S1	S0	BRKI	BITI	8086	FRQ
Command 1												Command 1							
PEN	EP	C1	C0	B3	B2	B1	B0	0	0	0	1	PEN	EP	C1	C0	B3	B2	B1	B0
Command 2												Command 2							
0	RxE	IAE	NIE	0	SBRK	TBRK	0	0	0	1	0	SET	RxE	IAE	NIE	END	SBRK	TBRK	RST
Command 3												Command 3							
T35	T24	T5C	CT3	CT2	P2C2	P2C1	P2C0	0	0	1	1	T35	T24	T5C	CT3	CT2	P2C2	P2C1	P2C0
Mode												Mode							
P17	P16	P15	P14	P13	P12	P11	P10	0	1	0	0	P17	P16	P15	P14	P13	P12	P11	P10
Port 1 Control												Port 1 Control							
L7	L6	L5	L4	L3	L2	L1	L0	0	1	0	1	L7	L6	L5	L4	L3	L2	L1	L0
Interrupt Enable												Set Interrupts							
D7	D6	D5	D4	D3	D2	D1	D0	0	1	1	0	L7	L6	L5	L4	L3	L2	L1	L0
Interrupt Address												Reset Interrupts							
D7	D6	D5	D4	D3	D2	D1	D0	0	1	1	1	D7	D6	D5	D4	D3	D2	D1	D0
Receiver Buffer												Transmitter Buffer							
D7	D6	D5	D4	D3	D2	D1	D0	1	0	0	0	D7	D6	D5	D4	D3	D2	D1	D0
Port 1												Port 1							
D7	D6	D5	D4	D3	D2	D1	D0	1	0	0	1	D7	D6	D5	D4	D3	D2	D1	D0
Port 2												Port 2							
D7	D6	D5	D4	D3	D2	D1	D0	1	0	1	0	D7	D6	D5	D4	D3	D2	D1	D0
Timer 1												Timer 1							
D7	D6	D5	D4	D3	D2	D1	D0	1	0	1	1	D7	D6	D5	D4	D3	D2	D1	D0
Timer 2												Timer 2							
D7	D6	D5	D4	D3	D2	D1	D0	1	1	0	0	D7	D6	D5	D4	D3	D2	D1	D0
Timer 3												Timer 3							
D7	D6	D5	D4	D3	D2	D1	D0	1	1	0	1	D7	D6	D5	D4	D3	D2	D1	D0
Timer 4												Timer 4							
D7	D6	D5	D4	D3	D2	D1	D0	1	1	1	0	D7	D6	D5	D4	D3	D2	D1	D0
Timer 5												Timer 5							
INT	RBF	TBE	TRE	BD	PE	OE	FE	1	1	1	1	0	RS4	RS3	RS2	RS1	RS0	TME	DSC
Status												Modification							

Break-In is polled by the MUART during the transmission of the last or only stop bit of a character.

A Break-In Detect is OR-ed with Break Detect in Bit 3 of the Status Register. The distinction can be made through the interrupt controller. If the transmit and receive interrupts are enabled, a Break-In will generate an interrupt on Level 5, the transmit interrupt, while Break will generate an interrupt on Level 4, the receive interrupt.

S0, S1 — Stop Bit Length

S1	S0	Stop Bit Length
0	0	1
0	1	1.5
1	0	2
1	1	0.75

The relationship of the number of stop bits and the function of input $\overline{CTS}$ is discussed in the Pin Description section under "$\overline{CTS}$".

L0, L1 — Character Length

L1	L0	Character Length
0	0	8
0	1	7
1	0	6
1	1	5

Command Register 2

PEN	EP	C1	C0	B3	B2	B1	B0
(1R)				(1W)			

Programming bits 0...3 with values from 3H to FH enables the internal baud rate generator as a common clock source for the transmitter and receiver and determines its divider ratio.

Programming bits 0...3 with values of 1H or 2H enables input TxC as a common clock source for the transmitter and receiver. The external clock must provide a frequency of either 32x or 64x the baud rate. The data transmission rates range from 0...32 Kbaud.

If bits 0...3 are set to 0, separate clocks must be input to pin RxC for the receiver and pin TxC for the transmitter. Thus, different baud rates can be used for transmission and reception. In this case, prescalers are disabled and the input serial clock frequency must match the baud rate. The input serial clock frequency can range from 0 to 1.024 MHz.

B0, B1, B2, B3 — Baud Rate Select

These four bits select the bit clock's source, sampling rate, and serial bit rate for the internal baud rate generator.

B3	B2	B1	B0	Baud Rate	Sampling Rate
0	0	0	0	$\overline{TxC}$, $\overline{RxC}$	1
0	0	0	1	$\overline{TxC}$/64	64
0	0	1	0	$\overline{TxC}$/32	32
0	0	1	1	19200	32
0	1	0	0	9600	64
0	1	0	1	4800	64
0	1	1	0	2400	64
0	1	1	1	1200	64
1	0	0	0	600	64
1	0	0	1	300	64
1	0	1	0	200	64
1	0	1	1	150	64
1	1	0	0	110	64
1	1	0	1	100	64
1	1	1	0	75	64
1	1	1	1	50	64

The following table gives an overview of the function of pins TxC and RxC:

Bits 3 to 0 (Hex.)	TxC	RxC
0	Input: 1 x baud rate clock for the transmitter	Input: 1 x baud rate clock for the receiver
1, 2	Input 32 x or 64 x baud rate for transmitter and receiver	Output: receiver bit clock with a low-to-high transition at data bit sampling time. Otherwise: high level
3 to F	Output: baud rate clock of the transmitter	Output: as above

As an output, RxC outputs a low-to-high transition at sampling time of evary data bit of a character. Thus, data can be loaded, e.g., into a shift register external-

ly. The transition occurs only if data bits of a character are present. It does not occur for start, parity, and stop bits (RxC = high).

As an output, TxC outputs the internal baud rate clock of the transmitter. There will be a high-to-low transition at every beginning of a bit.

C0, C1 — System Clock Prescaler (Bits 4, 5)

Bits 4 and 5 define the system clock prescaler divider ratio. The internal operating frequency of 1.024 MHz is derived from the system clock.

C1	C0	Divider Ratio	Clock at Pin CLK
0	0	5	5.12 MHz
0	1	3	3.072 MHz
1	0	2	2.048 MHz
1	1	1	1.024 MHz

EP — Even Parity (Bit 6)

EP = 0: Odd parity
EP = 1: Even parity

PEN — Parity Enable (Bit 7)

Bit 7 enables parity generation and checking.

PEN = 0: No parity bit
PEN = 1: Enable parity bit

The parity bit according to Command Register 2 bit 6 (see above) is inserted between the last data bit of a character and the first or only stop bit. The parity bit is checked during reception. A false parity bit generates an error indication in the Status Register and an Interrupt Request on Level 4.

Command Register 3

SET	RxE	IAE	NIE	END	SBRK	TBRK	RST
	(2R)				(2W)		

Command Register 3 is different from the first two registers because it has a bit set/reset capability. Writing a byte with Bit 7 high sets any bits which were also high. Writing a byte with Bit 7 low resets any bits which were high. If any bit 0-6 is low, no change occurs to that bit. When Command Register 3 is read, bits 0, 3, and 7 will always be zero.

RST — Reset

If RST is set, the following events occur:

1) All bits in the Status Register except bits 4 and 5 are cleared, and bits 4 and 5 are set.

2) The Interrupt Enable, Interrupt Request, and Interrupt Service Registers are cleared. Pending requests and indications for interrupts in service will be cancelled. Interrupt signal INT will go low.

3) The receiver and transmitter are reset. The transmitter goes idle (TxD is high), and the receiver enters start bit search mode.

4) If Port 2 is programmed for handshake mode, IBF and OBF are reset high.

RST does *not* alter ports, data registers or command registers, but it halts any operation in progress. RST is automatically cleared.

RST = 0 has no effect. The reset operation triggered by Command Register 3 is a subset of the hardware reset.

TBRK — Transmit Break

The transmission data output TxD will be set low as soon as the transmission of the previous character has been finished. It stays low until TBRK is cleared. The state of $\overline{CTS}$ is of no significance for this operation. As long as break is active, data transfer from the Transmitter Buffer to the Transmitter Register will be inhibited. As soon as TBRK is reset, the break condition will be deactivated and the transmitter will be reenabled.

SBRK — Single Character Break

This causes the transmitter data to be set low for one character including start bit, data bits, parity bit, and stop bits. SBRK is automatically cleared when time for the last data bit has passed. It will start after the character in progress completes, and will delay the next data transfer from the Transmitter Buffer to the Transmitter Register until TxD returns to an idle

(marking) state. If both TBRK and SBRK are set, break will be set as long as TBRK is set, but SBRK will be cleared after one character time of break. If SBRK is set again, it remains set for another character. The user can send a definite number of break characters in this manner by clearing TBRK after setting SBRK for the last character time.

END — End of Interrupt

If fully nested interrupt mode is selected, this bit resets the currently served interrupt level in the Interrupt Service Register. *This command must occur at the end of each interrupt service routine during fully nested interrupt mode.* END is automatically cleared when the Interrupt Service Register (internal) is cleared. END is ignored if nested interrupts are not enabled.

NIE — Nested Interrupt Enable

When NIE equals 1, the interrupt controller will operate in the nested interrupt mode. When NIE equals 0, the interrupt controller will operate in the normal interrupt mode. Refer to the "Interrupt controller" section under "Normal Mode" and "Nested Mode" for a detailed description of these operations.

IAE — Interrupt Acknowledge Enable

This bit enables an automatic response to $\overline{\text{INTA}}$. The particular response is determined by the 8086 bit in Command Register 1.

RxE — Receive Enable

This bit enables the serial receiver and its associated status bits in the status register. If this bit is reset, the serial receiver will be disabled and the receive status bits will not be updated.

Note that the detection of break characters remains enabled while the receiver is disabled; i.e., Status Register Bit 3 (BD) will be set while the receiver is disabled whenever a break character has been recognized at the receive data input RxD.

SET — Bit Set/Reset

If this bit is high during a write to Command Register 3, then any bit marked by a high will set. If this bit is low, then any bit marked by a high will be cleared.

Mode Register

T35	T24	T5C	CT3	CT2	P2C2	P2C1	P2C0
(3R)					(3W)		

P2C2, P2C1, P2C0 — Port 2 Control

P2C2	P2C1	P2C0	Mode	Direction Upper	Direction Lower
0	0	0	nibble	input	input
0	0	1	nibble	input	output
0	1	0	nibble	output	input
0	1	1	nibble	output	output
1	0	0	byte handshake	input	
1	0	1	byte handshake	output	
1	1	0	*DO NOT USE*		
1	1	1	test		

If test mode is selected, the output from the internal baud rate generator is placed on bit 4 of Port 1 (pin 35).

To achieve this, it is necessary to program bit 4 of Port 1 as an output (Port 1 Control Register Bit P14 = 1), and to program Command Register 2 bits B3 – B0 with a value ≥ 3H.

Note:
If Port 2 is operating in handshake mode, Interrupt Level 7 is not available for Timer 5. Instead it is assigned to Port 2 handshaking.

CT2, CT3 — Counter/Timer Mode

Bit 3 and 4 defines the mode of operation of event counter/timers 2 and 3 regardless of its use as a single unit or as a cascaded one.

If CT2 or CT3 are high, then counter/timer 2 or 3 respectively is configured as an event counter on bit 2 or 3 respectively of Port 1 (pins 37 or 36). The event counter decrements the count by one on each low-to-high transition of the external input. If CT2 or CT3 is low, then the respective counter/timer is configured as a timer and the Port 1 pins are used for parallel I/O.

T5C — Timer 5 Control

If T5C is set, then Timer 5 can be preset and started by an external signal. Writing to the Timer 5 register loads the Timer 5 save register and stops the timer. A high-to-low transition on bit 5 of Port 1 (pin 34) loads the timer with the saved value and starts the timer. The next high-to-low transition on pin 34 retriggers the timer by reloading it with the initial value and continues timing.

Following a hardware reset, the save register is reset to 00H and both clock and trigger inputs are disabled. Transferring an instruction with T5C = 1 enables the trigger input; the save register can now be loaded with

an initial value. The first trigger pulse causes the initial value to be loaded from the save register and enables the counter to count down to zero.

When the timer reaches zero it issues an interrupt request, disables its interrupt level and continues counting. A subsequent high-to-low transition on pin 5 resets Timer 5 to its initial value. For another timer interrupt, the Timer 5 interrupt enable bit must be set again.

T35, T24 — Cascade Timers

These two bits cascade Timers 3 and 5 or 2 and 4. Timers 2 and 3 are the lower bytes, while Timers 4 and 5 are the upper bytes. If T5C is set, then both Timers 3 and 5 can be preset and started by an external pulse. When a high-to-low transition occurs, Timer 5 is preset to its saved value, But Timer 3 is always preset to all ones. If either CT2 or CT3 is set, then the corresponding timer pair is a 16-bit event counter.

A summary of the counter/timer control bits is given in Table 5.

Note:
Interrupt levels assigned to single counters are partly not occupied if event counters/timers are cascaded. Level 2 will be vacated if event counters/timers 2 and 4 are cascaded. Likewise, Level 7 will be vacated if event counters/timers 3 and 5 are cascaded.

Single event counters/timers generate an interrupt request on the transition from 01H to 00H, while cascaded ones generate it on the transition from 0001H to 0000H.

Table 5. Event Counters/Timers Mode of Operation

Event Counter/ Timer	Function	Programming (Mode Word)	Clock Source
1	8-bit timer	–	internal clock
2	8-bit timer	T24 = 0, CT2 = 0	internal clock
	8-bit event counter	T24 = 0, CT2 = 1	P12 pin 37
3	8-bit timer	T35 = 0, CT3 = 0	internal clock
	8-bit event counter	T35 = 0, CT3 = 1	P13 pin 36
4	8-bit timer	T24 = 0	internal clock
5	8-bit timer, normal mode	T35 = 0, T5C = 0	internal clock
	8-bit timer, retriggerable mode	T35 = 0, T5C = 1	internal clock
2 and 4 cascaded	16-bit timer	T24 = 1, CT2 = 0	internal clock
	16-bit event counter	T24 = 1, CT2 = 1	P12 pin 37
3 and 5 cascaded	16-bit timer, normal mode	T35 = 1, T5C = 0, CT3 = 0	internal clock
	16-bit event counter, normal mode	T35 = 1, T5C = 0, CT3 = 1	P13 pin 36
	16-bit timer, Retriggerable mode	T35 = 1, T5C = 1, CT3 = 0	internal clock
	16-bit event counter, Retriggerable mode	T35 = 1, T5C = 1, CT3 = 1	P13 pin 36

Port 1 Control Register

P17	P16	P15	P14	P13	P12	P11	P10
(4R)				(4W)			

Each bit in the Port 1 Control Register configures the direction of the corresponding pin. If the bit is high, the pin is an output, and if it is low the pin is an input. Every Port 1 pin has another function which is controlled by other registers. If that special function is disabled, the pin functions as a general I/O pin as specified by this register. The special functions for each pin are described below.

Port 10, 11 — Handshake Control

If byte handshake control is enabled for Port 2 by the Mode Register, then Port 10 is programmed as $\overline{STB}/\overline{ACK}$ handshake control input, and Port 11 is programmed as $\overline{IBF}/\overline{OBF}$ handshake control output.

If byte handshake mode is enabled for output on Port 2, $\overline{OBF}$ indicates that a character has been loaded into the Port 2 output buffer. When an external device reads the data, it acknowledges this operation by driving $\overline{ACK}$ low. $\overline{OBF}$ is set low by writing to Port 2 and is reset high by $\overline{ACK}$.

If byte handshake mode is enabled for input on Port 2, $\overline{STB}$ is an input. $\overline{IBF}$ is driven low after $\overline{STB}$ goes low. On the rising edge of $\overline{STB}$ the data from Port 2 is latched.

$\overline{IBF}$ is reset high when Port 2 is read.

Port 12, 13 — Counter 2, 3 Input

If Timer 2 or Timer 3 is programmed as an event counter by the Mode Register, then Port 12 or Port 13 is the counter input for Event Counter 2 or 3, respectively.

Port 14 — Baud Rate Generator Output Clock

If test mode is enabled by the Mode Register and Command Register 2 baud rate select is greater than 2, then Port 14 is an output from the internal baud rate generator.

P14 in Port 1 control register must be set to 1 for the baud rate generator clock to be output. The baud rate generator clock is 64 x the serial bit rate except at 19.2Kbps when it is 32 x the bit rate.

Port 15 — Timer 5 Trigger

If T5C is set in the Mode Register enabling a retriggerable timer, then Port 15 is the input which starts and reloads Timer 5.

A high-to-low transition on P15 (Pin 34) loads the timer with the save register and starts the timer.

Port 16 — Break-In Detect

If Break-In Detect is enabled by BRKI in Command Register 1, then this input is used to sense a Break-In. If Port 16 is low while the serial transmitter is sending the last stop bit, then a Break-In condition is signaled.

Port 17 — Port Interrupt Source

If BITI in Command Register 1 is set, then a low-to-high transition on Port 17 generates an interrupt request on Priority Level 1.

Port 17 is edge triggered.

Interrupt Enable Register

L7	L6	L5	L4	L3	L2	L1	L0
	(5R)			(5W = enable, 6W = disable)			

Interrupts are enabled by writing to the Set Interrupts Register (5W). Interrupts are disabled by writing to the Reset Interrupts Register (6W). Each bit set by the Set Interrupts Register (5W) will enable that level interrupt, and each bit set in the Reset Interrupts Register (6W) will disable that level interrupt. The user can determine which interrupts are enabled by reading the Interrupt Enable Register (5R).

Priority		Source
Highest	L0	Timer 1
	L1	Timer 2 or Port Interrupt
	L2	External Interrupt (EXTINT)
	L3	Timer 3 or Timers 3 & 5
	L4	Receiver Interrupt
	L5	Transmitter Interrupt
	L6	Timer 4 or Timers 2 & 4
Lowest	L7	Timer 5 or Port 2 Handshaking

Interrupt Address Register

0	0	0	D4	D3	D2	0	0
(6R)					Interrupt Level Indication		

Reading the interrupt address register transfers an identifier for the currently requested interrupt level on the system data bus. This identifier is the number of the interrupt level multiplied by 4. It can be used by the CPU as an offset address for interrupt handling. Reading the interrupt address register has the same effect as a hardware interrupt acknowledge $\overline{INTA}$; it clears the interrupt request pin (INT) and indicates an interrupt acknowledgement to the interrupt controller.

Receiver and Transmitter Buffer

D7	D6	D5	D4	D3	D2	D1	D0
(7R)				(7W)			

Both the receiver and transmitter in the MUART are double buffered. This means that the transmitter and receiver have a shift register and a buffer register. The buffer registers are directly addressable by reading or writing to register seven. After the receiver buffer is full, the RBF bit in the status register is set. Reading the receive buffer clears the RBF status bit. The transmit buffer should be written to only if the TBE bit in the status register is set. Bytes written to the transmit buffer are held there until the transmit shift register is empty, assuming $\overline{CTS}$ is low. If the transmit buffer and shift register are empty, writing to the transmit buffer immediately transfers the byte to the transmit shift register. If a serial character length is less than 8 bits, the unused most significant bits are set to zero when reading the receive buffer, and are ignored when writing to the transmit buffer.

Port 1

D7	D6	D5	D4	D3	D2	D1	D0
(8R)				(8W)			

Writing to Port 1 sets the data in the Port 1 output latch. Writing to an input pin does not affect the pin, but the data is stored and will be output if the direction of the pin is changed later. If the pin is used as a control signal, the pin will not be affected, but the data is stored. Reading Port 1 transfers the data in Port 1 onto the data bus.

Port 2

D7	D6	D5	D4	D3	D2	D1	D0
	(9R)				(9W)		

Writing to Port 2 sets the data in the Port 2 output latch. Writing to an input pin does not affect the pin, but it does store the data in the latch. Reading Port 2 puts the input pins onto the bus or the contents of the output latch for output pins.

Timer 1-5

D7	D6	D5	D4	D3	D2	D1	D0
	($0A_{16}$-$0E_{16}$R)				($0A_{16}$-$0E_{16}$W)		

Reading Timer N puts the contents of the timer onto the data bus. If the counter changes while $\overline{RD}$ is low, the value on the data bus will not change. If two timers are cascaded, reading the high-order byte will cause the low-order byte to be latched. Reading the low-order byte will unlatch them both. Writing to either timer or decascading them also clears the latch condition. Writing to a timer sets the starting value of that timer. If two timers are cascaded. writing to the high-order byte presets the low-order byte to all ones. Loading only the high-order byte with a value of X leads to a count of X 256+255. Timers count down continuously. If the interrupt is enabled, it occurs when the counter changes from 1 to 0.

The timer/counter interrupts are automatically disabled when the interrupt request is generated.

Status Register

INT	RBF	TBE	TRE	BD	PE	OE	FE
			($0F_{16}$R)				

Reading the status register gates its contents onto the data bus. It holds the operational status of the serial interface as well as the status of the interrupt pin INT. The status register can be read at any time. The flags are stable and well defined at all instants.

FE — Framing Error, Transmission Mode

Bit 0 can be used in two modes. Normally, FE indicates framing error which can be changed to transmission mode indication by setting the TME bit in the modification register.

If transmission mode is disabled (in Modification Register), then FE indicates a framing error. A framing error is detected during the *first* stop bit. The error is reset by reading the Status Register or by a chip reset. A framing error does not inhibit the loading of the Receiver Buffer. If RxD remains low, the receiver will assemble the next character. The false stop bit is treated as the next start bit, and no high-to-low transition on RxD is requied to synchronize the receiver.

When the TME bit in the Modification Register is set, FE is used to indicate that the transmitter was active during the reception of a character, thus indicating that the character received was transmitted by its own transmitter. FE is reset when the transmitter is not active during the reception of character. Reading the status register will not reset the FE bit in the transmission mode.

OE — Overrun Error

If the user does not read the character in the Receiver Buffer before the next character is received and transferred to this register, then the OE bit is set. The OE flag is set during the reception of the first stop bit and is cleared when the Status Register is read or when a hardware or software reset occurs. The first character received in this case will be lost.

PE — Parity Error

This bit indicates that a parity error has occurred during the reception of a character. A parity error is present if value of the parity bit in the received character is different from the one expected according to command word 2 bits 6 EP. The parity bit is expected and checked only if it is enabled by command word 2 bit 7 PEN.

A parity error is set during the first stop bit and is reset by reading the Status Register or by a chip reset.

BD — Break/Break-In

The BD bit flags whether a break character has been received, or a Break-In condition exists on the transmission line. Command Register 1 Bit 3 (BRKI) enables the Break-In Detect function.

Whenever a break character has been received, Status Register Bit 3 will be set and in addition an interrupt request on Level 4 is generated. The receiver will be idled. It will be started again with the next high-to-low transition at pin RxD.

The break character received will not be loaded into the receiver buffer register.

If Break-In Detection is enabled and a Break-In condition occurs, Status Register Bit 3 will be set and in addition an interrupt request on Level 5 is generated.

The BD status bit will be reset on reading the status register or on a hardware or software reset. For more information on Break/Break-In, refer to the "Serial Asynchronous Communication" section under "Receive Break Detect"and "Break-In Detect."

TRE — Transmit Register Empty

When TRE is set the transmit register is empty and an interrupt request is generated on Level 5 if enabled. When TRE equals 0 the transmit register is in the process of sending data. TRE is set by a chip reset and when the last stop bit has left the transmitter. It is reset when a character is loaded into the Transmitter Register. If $\overline{CTS}$ is low, the Transmitter Register will be loaded during the transmission of the start bit. If $\overline{CTS}$ is high at the end of a character, TRE will remain high and no character will be loaded into the Transmitter Register until $\overline{CTS}$ goes low. If the transmitter was inactive before a character is loaded into the Transmitter Buffer, the Transmitter Register will be empty temporarily while the buffer is full. However, the data in the buffer will be transferred to the transmitter register immediately and TRE will be cleared while TBE is set.

TBE — Transmitter Buffer Empty

TBE indicates the Transmitter Buffer is empty and is ready to accept a character. TBE is set by a chip reset or the transfer of data to the Transmitter Register, and is cleared when a character is written to the transmitter buffer. When TBE is set, an interrupt request is generated on Level 5 if enabled.

RBF — Receiver Buffer Full

RBF is set when the Receiver Buffer has been loaded with a new character during the sampling of the first stop bit. RBF is cleared by reading the receiver buffer or by a chip reset.

INT — Interrupt Pending

The INT bit reflects the state of the INT Pin (Pin 15) and indicates an interrupt is pending. It is reset by $\overline{INTA}$ or by reading the Interrupt Address Register if only one interrupt is pending and by a chip reset.

FE, OE, PE, RBF, and Break Detect all generate a Level 4 interrupt when the receiver samples the first stop bit. TRE, TBE, and Break-In Detect generate a Level 5 interrupt. TRE generates an interrupt when TBE is set and the Transmitter Register finished transmitting. The Break-In Detect interrupt is issued at the same time as TBE or TRE.

Modification Register

0	RS4	RS3	RS2	RS1	RS0	TME	DSC

($0F_{16}$W)

DSC — Disable Start Bit Check

DSC disables the receiver's start bit check. In this state the receiver will not be reset if RxD is not low at the center of the start bit.

TME — Transmission Mode Enable

TME enables transmission mode and disables framing error detection. For information on transmission mode see the description of the framing error bit in the Status Register.

RS0, RS1, RS2, RS3, RS4 — Receiver Sample Time

The number in RSn alters when the receiver samples RxD. The receiver sample time can be modified only if the receiver is *not* clocked by RxC.

Note:
The modification register cannot be read. Reading from address 0FH, 8086: 1EH gates the contents of the status register onto the data bus.

- A hardware reset (reset, Pin 12) resets all modification register bits to 0, i.e.:
  * The start bit check is enabled.
  * Status Register Bit 0 (FE) indicates framing error.
  * The sampling time of the serial receiver is the bit center.

  A software reset (Command Word 3, RST) does not affect the modification register.

Hardware Reset

A reset signal on pin RESET (HIGH level) forces the device 8256 into a well-defined initial state. This state is characterized as follows:

RS4	RS3	RS2	RS1	RS0	Point of time between start of bit and end of bit measured in steps of 1/32 bit length	
0	1	1	1	1	1	(Start of Bit)
0	1	1	1	0	2	
0	1	1	0	1	3	
0	1	1	0	0	4	
0	1	0	1	1	5	
0	1	0	1	0	6	
0	1	0	0	1	7	
0	1	0	0	0	8	
0	0	1	1	1	9	
0	0	1	1	0	10	
0	0	1	0	1	11	
0	0	1	0	0	12	
0	0	0	1	1	13	
0	0	0	1	0	14	
0	0	0	0	1	15	
0	0	0	0	0	16	(Bit center)
1	1	1	1	1	17	
1	1	1	1	0	18	
1	1	1	0	1	19	
1	1	1	0	0	20	
1	1	0	1	1	21	
1	1	0	1	0	22	
1	1	0	0	1	23	
1	1	0	0	0	24	
1	0	1	1	1	25	
1	0	1	1	0	26	
1	0	1	0	1	27	
1	0	1	0	0	28	
1	0	0	1	1	29	
1	0	0	1	0	30	
1	0	0	0	1	31	
1	0	0	0	0	32	(End of Bit)

1) Command registers 1, 2 and 3, mode register, Port 1 control register, and modification register are reset. Thus, all bits of the parallel interface are set to be inputs and event counters/timers are configured as independent 8-bit timers.

2) Status register bits are reset with the exception of bits 4 and 5. Bits 4 and 5 are set indicating that both transmitter register and transmitter buffer register are empty.

3) The interrupt mask, interrupt request, and interrupt service register bits are reset and disable all requests. As a consequence, interrupt signal INT is inactive (LOW).

4) The transmit data output is set to the marking state (HIGH) and the receiver section is disabled until it is enabled by Command Register 3 Bit 6.

5) The start bit will be checked at sampling time. The receiver will return to start bit search mode if input RxD is not LOW at this time.

6) Status Register Bit 0 implies framing error.

7) The receiver samples input RxD at bit center.

Reset has no effect on the contents of receiver buffer register, transmitter buffer register, the intermediate latches of parallel ports, and event counters/timers, respectively.

INTERFACING

This section describes the hardware interface between the 8256 MUART and the 8085, 8086, 8088, and 80186 microprocesors. Figures 14 through 19 display the block diagrams for these interfaces. The MUART can be interfaced to many other microprocessors using these basic principles.

In all cases the 8256 will be connected directly to the CPU's multiplexed address/data bus. If latches or data bus buffers are used in a system, the MUART should be on the microprocessor side of the address/data bus. The MUART latches the address internally on the falling edge of ALE. The address consists of Chip Select (CS) and four address lines. For 8-bit microprocessors, AD0-AD3 are the address lines. For 16-bit microprocessors, AD1-AD4 are the address lines; AD0 is used as a second chip select which is active low. Since chip select is internally latched along with the address, it does not have to remain active during the entire instruction cycle. As long as the chip select setup and hold times are met, it can be derived from multiplexed address/data lines or multiplexed address/status lines.

In Figure 15, the 8088 min mode, the 8205 chip select decoder is connected to the 8088's address bus lines A8-A15. These address lines are stable throughout the entire instruction cycle. However, the MUART's chip select signal could have been derived from A16/S3-A19/S6.

Figure 16 shows the 8256 interfaced with an 8086 in the min mode. When the 8256 is in the 16-bit mode, A0 serves as a second chip select. As a result the MUART's internal registers will all have even addresses since A0 must be zero to select the device. Normally the MUART will be placed on the lower data byte. If the MUART is placed on the upper data byte the internal registers will be 512 address locations apart and the chip would occupy an 8 K word address space. Figure 16A shows a table and a diagram of how the 8256 may be selected in an 8086 system where the MUART is I/O mapped and used on the lower byte of the address/data bus.

PROGRAMMING

Initialization

In general the MUART's functions are independent of each other and only the registers and bits associated with a particular function need to be initialized, not the entire chip. The command sequence is arbitrary since every register is directly addressable; however, Command Word 1 must be loaded first. To put the device into a fully operational condition, it is necessary to write the following commands:

Command byte 1
Command byte 2
Command byte 3
Mode byte
Port 1 control
Set Interrupts

The modification register may be loaded if required for special applications; normally this operation is not necessary. It is a good idea to reset the part before initialization. (Either a hardware or a software reset will do.)

Operating the Serial Interface

The microprocessor transfers data to the serial interface by writing bytes to the Transmit Buffer Register. Receive characters are transferred by reading the Receiver Buffer Register. The Status Register provides all of the necessary information to operate the serial I/O, including when to write to the Transmit Buffer, and when to read the Receive Buffer and error information.

Transmitting

The transmitter and the receiver may be operated by using either polling or interrupts. If polling is used then the software may poll the Status Register and write a byte to the Transmit Buffer whenever TBE = 1. Writing a byte to the Transmit Buffer clears the TBE

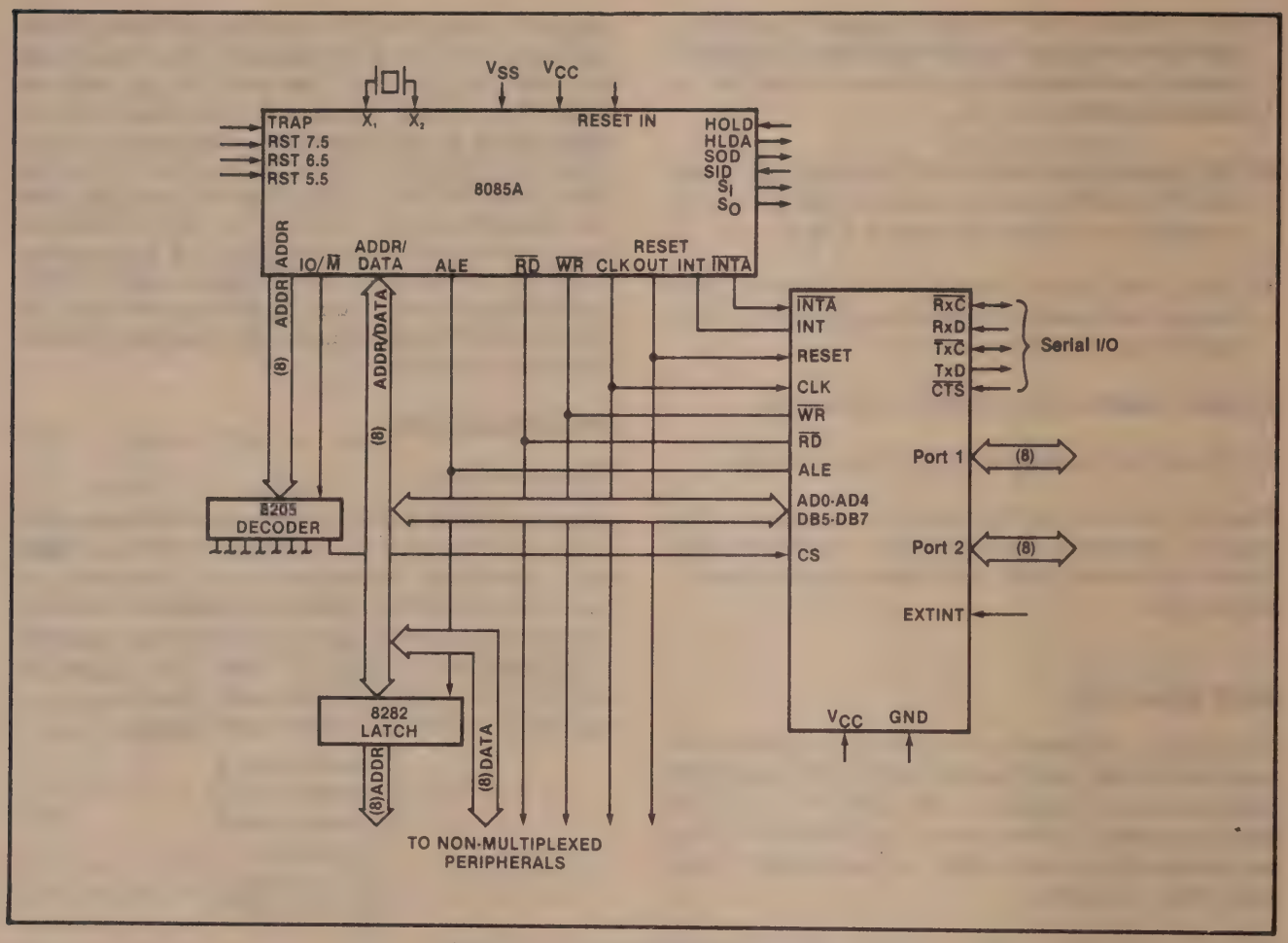


Figure 14. 8085/8256 Interface

status bit. If the $\overline{CTS}$ pin is low, then the Transmit Buffer will transfer the data to the Transmit Register when it becomes empty. When this transfer takes place the TRE bit is reset, and the TBE bit is set indicating the next byte may be written to the Transmit Buffer. If $\overline{CTS}$ is high, disabling the transmitter, the data byte will remain in the Transmit Buffer and TBE will remain low until $\overline{CTS}$ goes low. The transmitter can only buffer one byte if it is disabled.

There is no way of knowing that the transmitter is disabled unless the $\overline{CTS}$ signal is fed into one of the I/O ports. Using the transmitter interrupt will free up the CPU to perform other functions while the transmitter is disabled or while the Transmit Buffer is full.

To enable the transmit interrupt feature Bit L5 in the Set Interrupt Register must be set. An interrupt request will not occur immediately after this bit has been set. Before any transmit interrupt request will occur a byte must be written to the Transmit Buffer. After the first byte has been written to the Transmit Buffer, a transmit interrupt request will occur, providing the transmitter is enabled.

There are three sources of transmitter interrupt requests: TBE = 1, TRE = 1, and Break-In Detect. Assuming the Break-In Detect feature is disabled, after the transmit interrupt is enabled and the first byte is written, a transmit interrupt request will be generated by TBE going active. The microprocessor can immediately write a byte to the Transmit Buffer without reading any status. However if Break-In Detect is enabled, the Status Register must be read to determine whether the transmit interrupt request was generated by Break-In Detect or TBE.

The TRE interrupt request can be used to indicate when the transmitter has completely sent all of the data. For example, using half-duplex communica-

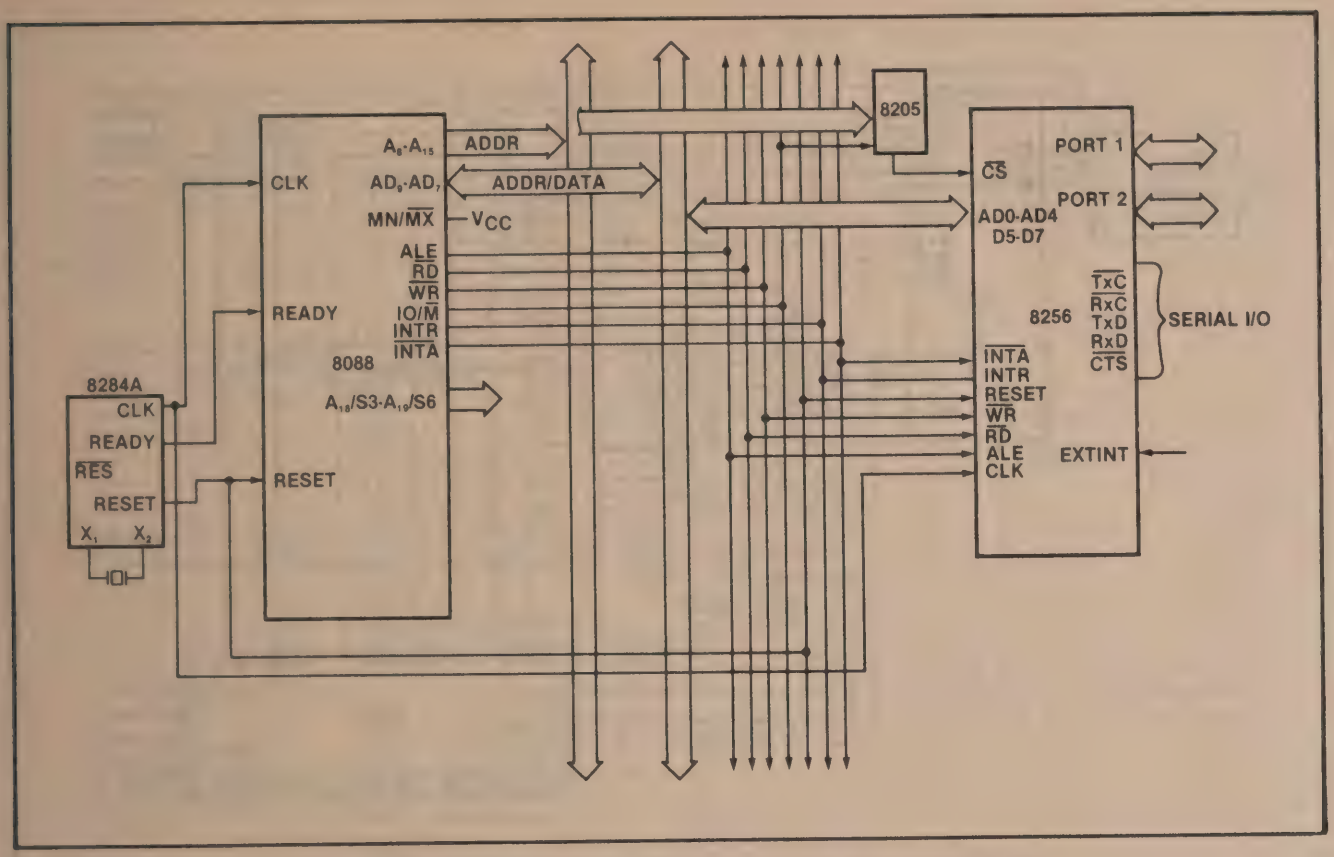


Figure 15. 8088 Min Mode/8256 Interface Multiplexed Bus

tions, all of the data written to the MUART must be transmitted before the line can be turned around. After the last byte is written, an interrupt request will be generated by TBE. If this interrupt is acknowledged without writing another byte, then the next transmitter interrupt request, TRE = 1, will indicate that the transmitter is empty and the line may be turned around.

RECEIVING

Valid data may be read from the Receive Buffer whenever the RBF bit in the Status Register is set. Reading the Receive Buffer resets the RBF status bit. The RBF bit in the Status Register can be used for polling. When the RBF bit is set, the three receive status bits, PE, OE, and FE are updated. These three status bits are reset when they are read. Therefore when the status register is read with RBF set, the three error status bit should be tested too.

If interrupts are used for serial receive data, the receiver must be enabled by setting the RxE bit in Command Register 3, and Bit L4 must be set in the Set Interrupt Register. When the receive interrupt request occurs the Receive Buffer may be read, but the status register should also be read since the receive interrupt could have been generated by the Break Detect. Also, reading the status register will indicate whether there were any errors in the received character.

Operating the Parallel Interface

Data can be transferred to or read from Port 1 and Port 2 by using the appropriate write and read operations.

LOADING PORT 1 and PORT 2

Writing to the ports transfers the data present on the data bus into the output latches. This operation is independent of the programmed I/O characteristics of the individual port pins. Writing to control or input ports has no effect on the state of the pins. Pins defined as outputs immediately assume the state which is associated with the transferred data. If inputs or control pins are reprogrammed into outputs, they assume the states stored in their output latches which were transferred by the most recent port write operation.

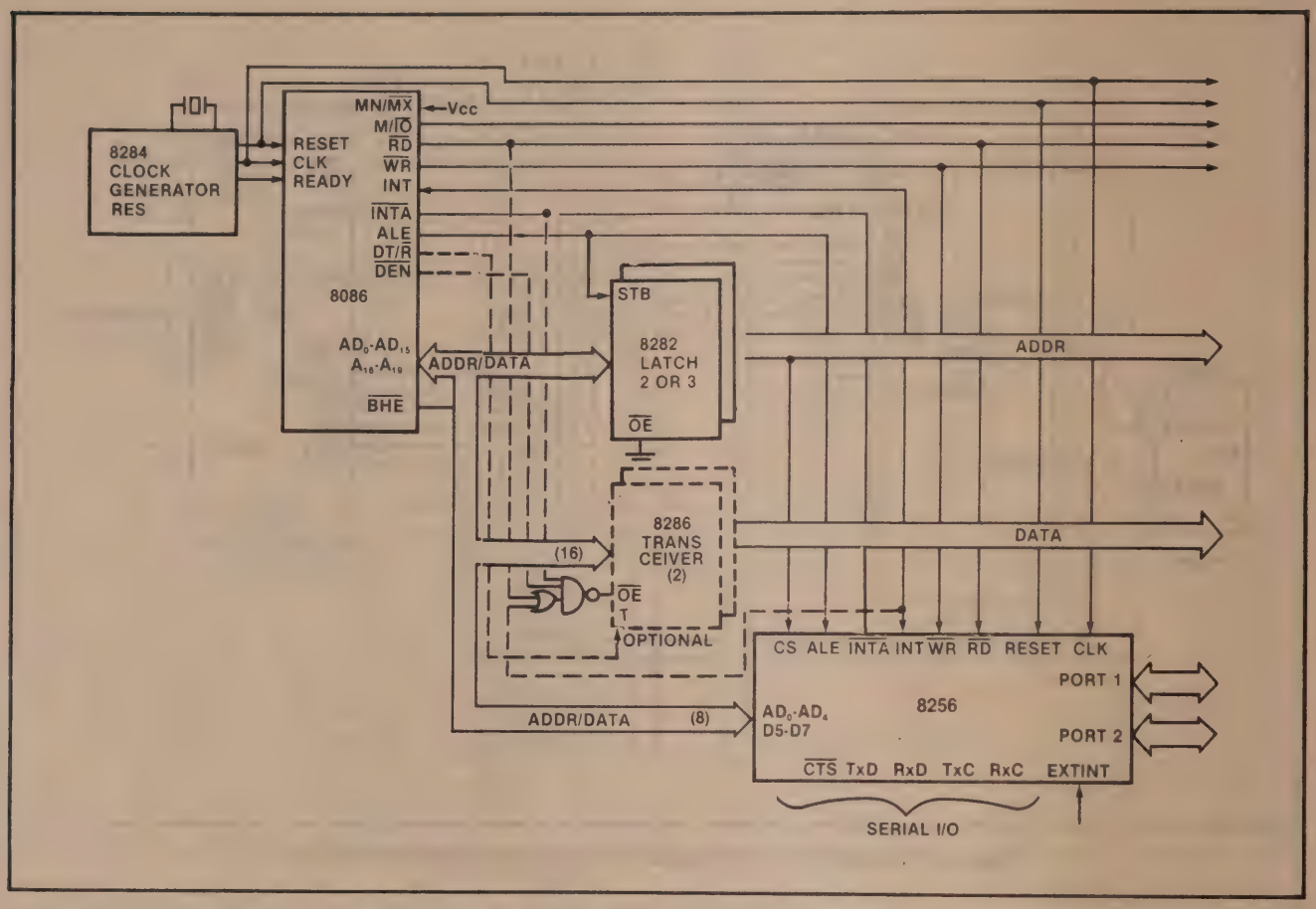


Figure 16. 8086 Min Mode/8256 Interface

	$\overline{BHE}$	A_0	CHARACTERISTICS
	0	0	WHOLE WORD
	0	1	UPPER BYTE FROM/TO ODD ADDRESS
MUART→	1	0	LOWER BYTE FROM/TO EVEN ADDRESS
	1	1	NONE

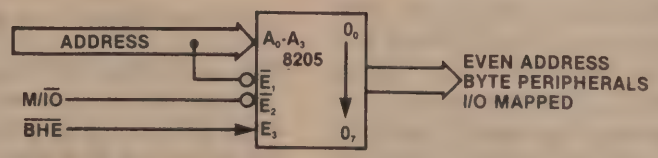


Figure 16a. Technique for Generating the MUART's Chip Select

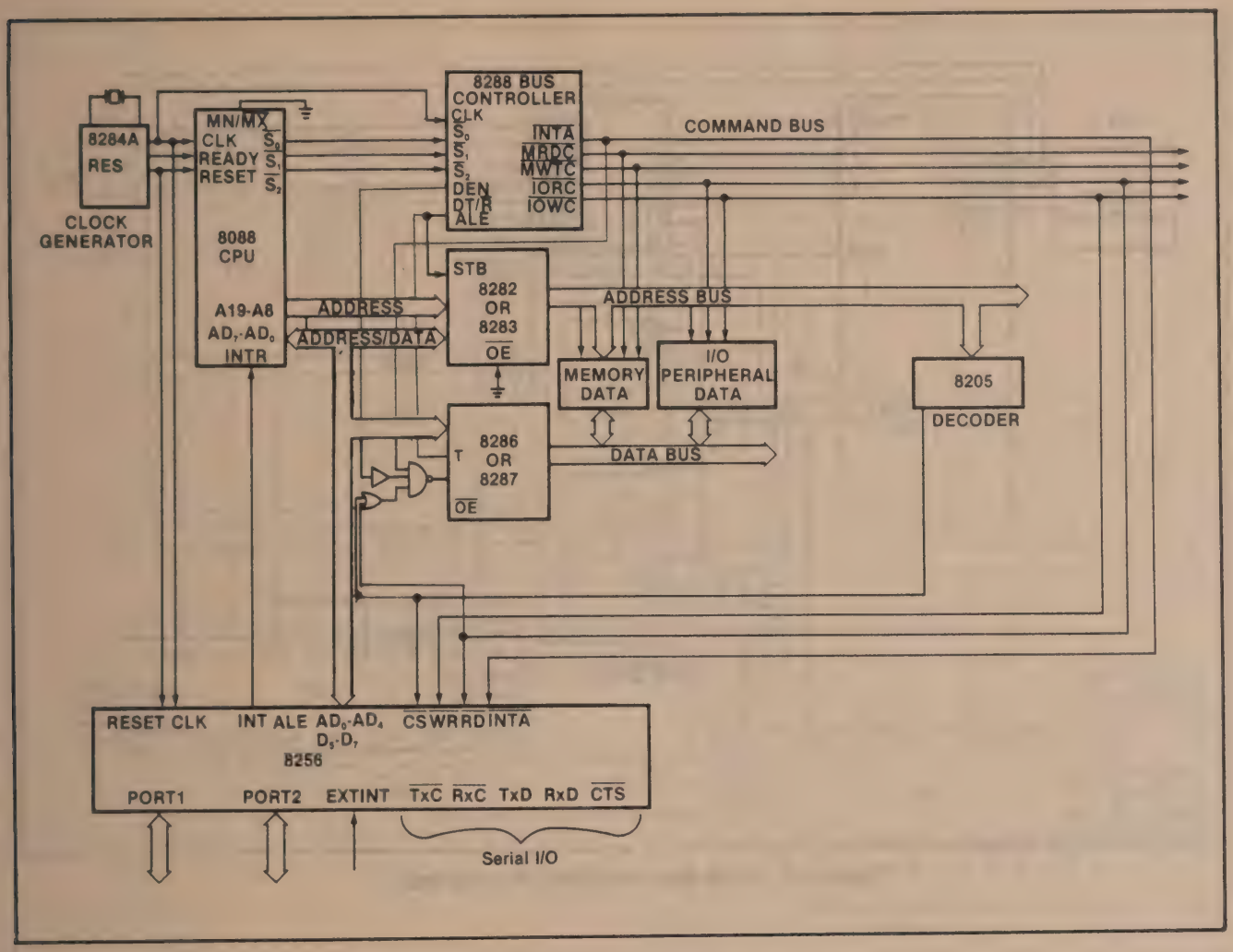


Figure 17. 8088 Max Mode/8256 Interface

READING PORT 1 AND PORT 2

Reading the ports gates the state at the pins onto the data bus if they are defined as I/O pins. A read operation transfers the contents of the associated output latches of pins P12, P13, P15, and P16, which are defined as control function pins. Reading control pins P10, P11, and P17 delivers the state of these pins.

Operating the Event Counters/Timers

The event counters/timers can be loaded with an initial value at any time. Reading event counters/timers is possible without interfering with the counting process.

LOADING EVENT COUNTERS/TIMERS

Loading event counters/timers 1-5 under their respective addresses transfers the data present on the data bus as an initial value into the addressed event counter/timer. The event counter/timer counts from the new initial value immediately following the data transfer (exception: retriggerable mode of Timer 5, or 3 and 5)

Cascaded counters/timers can be loaded with an initial value using one of two procedures:
1) Only the event counter/timer representing the most significant byte will be loaded. The event counter/timer representing the least significant byte is set to 0FFH automatically. Counting is started immediately after the data transfer.
2) The event counter/timer representing the most significant byte will be loaded, causing the least significant byte to be set to 0FFH automatically. Counting is started immediately following the data transfer. Next, the counter representing the least significant byte will be loaded and counting is started

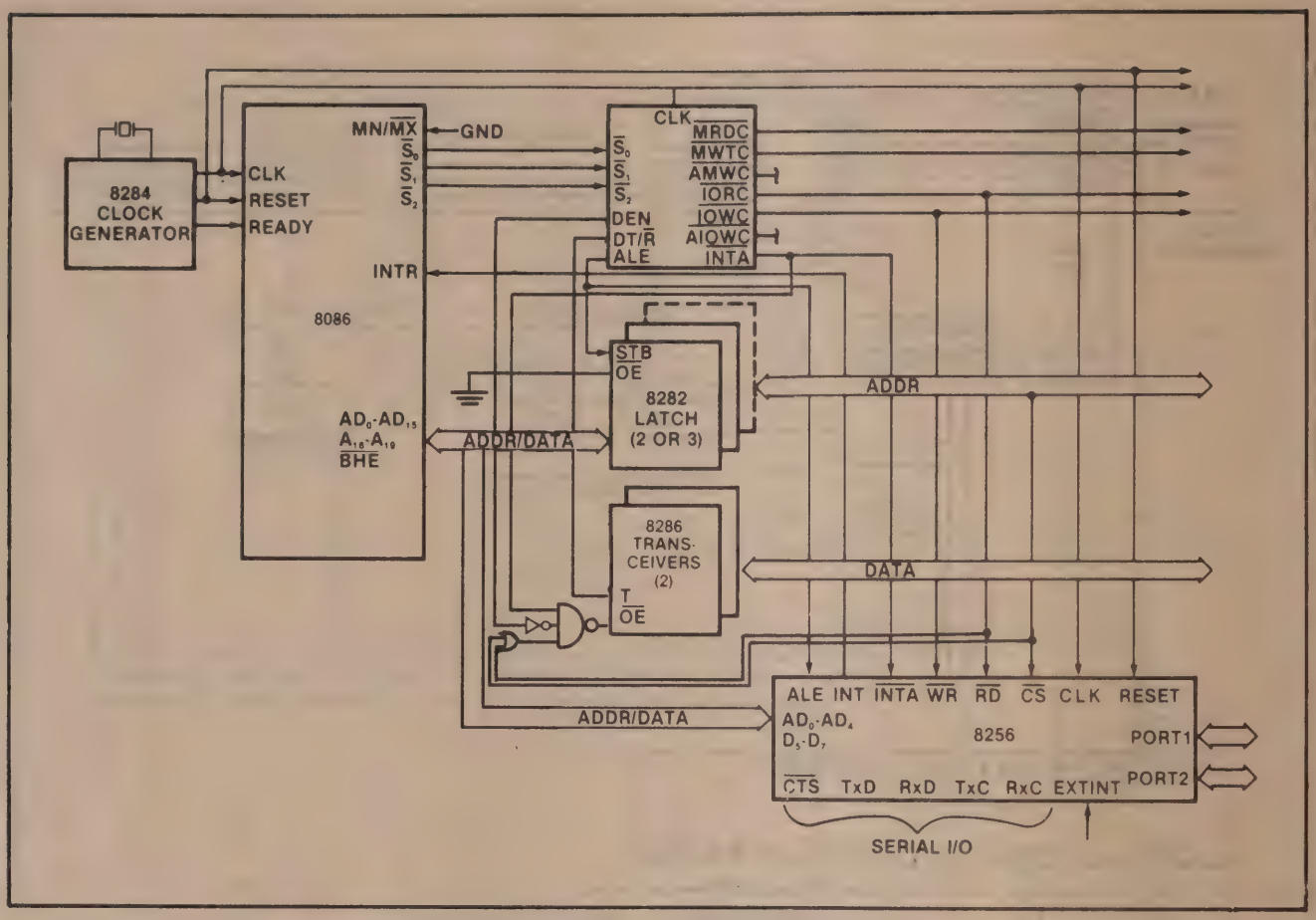


Figure 18. 8086 Max Mode/8256 Interface

again, but this time with a complete 16-bit initial value. The least significant byte of the initial value must be transferred before the counter representing the least significant byte exhibits its zero transition to prevent the most significant byte of the initial value from being decremented improperly.

In the case of an 8-bit initial value for Timer 5 or for cascaded Event Counter/Timer 3 and 5, the initial value for Timer 5 is loaded from a save register, if it is operated in retriggerable counting mode. Counting is started after an initial value has been transferred whenever a high-to-low transition occurs on Port P15.

Cascaded Event Counter/Timer 3 and 5 operating in retriggerable counting mode can be loaded directly with an initial value for Timer 5 representing the most significant byte; Event Counter/Timer 3 will be set to 0FFH automatically.

READING EVENT COUNTERS/TIMERS

Reading event counters/timers 1-5 from their respective addresses gates the counter contents onto the data bus. The counter contents gated onto the data bus remain stable during the read operation while the counter just being read continues to count. The minimum time between the two read operations from the same counter is 1 usec.

The procedure to be followed when reading cascaded event counters/timers is:
1) The event counter/timer representing the most significant byte will be read first. At this time, the least significant byte is latched into read latches.
2) When the event counter/timer representing the least significant byte is addressed, the byte stored in the read latches will be gated onto the data bus. The value stored in the read latches remains valid until it is read, the cascading condition is removed, or a write

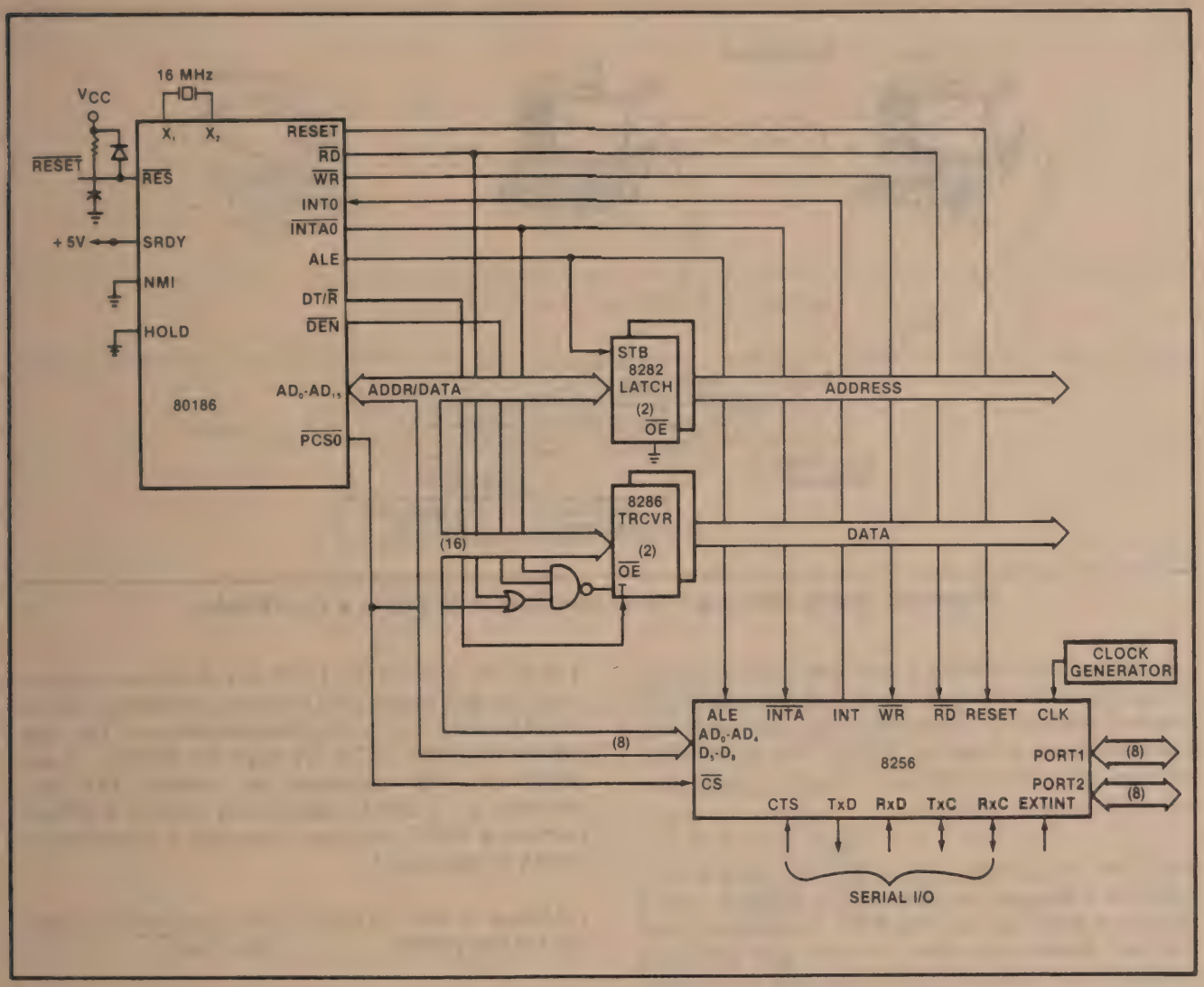


Figure 19. 80186/8256 Interface

operation affecting one of the two event counters/timers is executed.

The time between reading the most significant byte and the least significant byte must be at least 1 usec.

Note:
For cascaded event counters/timers the least significant counter/timer is latched after reading the most significant counter/timer. If the lower byte changes from 00H to 0FFH between the reading of the MSB and the latching of the LSB, the carry from the most significant event counter/timer to the least significant event counter/timer is lost.

Therefore, it is necessary to repeat the whole reading once if the value of the least significant event counter/timer is 0FFH. Doing this will avoid working with a wrong value (correct value + 255).

APPLICATION EXAMPLE

This section describes how the 8256 was designed into a Line Printer Multiplexer (LPM). This application example was chosen because it employs a majority of the MUART's features. The information in this section will be applicable to many other designs since it describes some common software and hardware aspects of using the MUART.

Description of the Line Printer Multiplexer (LPM)

The Line Printer Multiplexer allows up to eight workstations to share one printer. The workstations transmit serial asynchronous data to the LPM. The LPM receives the serial data, buffers it, then transmits

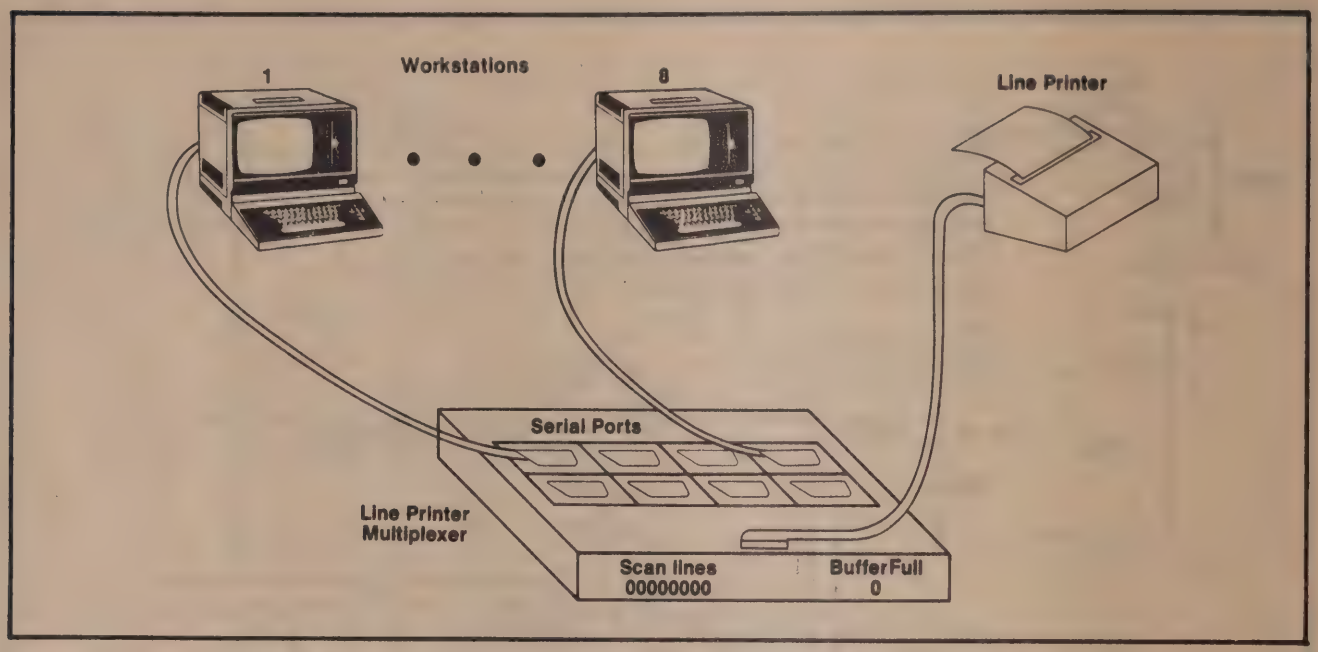


Figure 20. Using the Line Printer Multiplexer to Share a Line Printer

it to the line printer using a two-wire byte handshake Dataproducts interface. A conceptual diagram of this system is shown in Figure 20. Note that only one workstation can transmit at a time. This workstation will transmit its entire file before another workstation will be allowed to transmit.

The LPM sequentially polls each of the eight RS-232 ports for a Request To Send ($\overline{RTS}$). When it finds a serial port which has asserted $\overline{RTS}$, it configures itself for the appropriate data format and bit rate, establishes the connection and sends back to the serial port a Clear To Send ($\overline{CTS}$) which enables transmission. The LPM receives the serial asynchronous data, buffers it in a software FIFO, and transmits the data to the line printer. If the LPM detects an error in any of the serial characters it receives, it transmits an error message to the serial port and ignores the bad character. If the LPM does not receive a serial character after 18 seconds, it assumes that the transmission is complete. It transmits the final status to the serial port, and returns to scanning.

This LPM was designed to be used with single-user workstations and a 300 lines per minute line printer. These workstations are not multitasking; therefore in the middle of a file transfer when the CPU needs to reload its buffer from the disk, no serial data is transmitted. During this time the LPM is emptying its FIFO; thus, the line printer never stops printing.

The buffer size on the LPM was chosen to complement the disk access time on the workstations. Figure 21 illustrates the buffer size calculation. The line printer can print up to 300 lines per minute, or approximately 660 characters per second. This corresponds to a serial transmission rate of 6,600bps (assuming ASCII character codes and a parity bit) as shown in equation 1.

(1) Serial bit rate for the line printer $= \dfrac{(300\ \text{lines/min})*(132\ \text{char/line})*(10\ \text{bits/char})}{(60\ \text{sec/min})}$

The bottleneck in this data transfer is the line printer since the MUART and the workstations can both transmit and receive at 19.2Kbps. To realize the maximum data transfer rate of this system the LPM must guarantee that the average transfer rate to the line printer is 660 characters per second. The maximum amount of dead time that the serial port on the workstation is not transmitting, multiplied by 660 is the number of bytes which the LPM should buffer. It was determined through experimentation that it takes about 3 seconds to load 40K bytes of data from the disk into the workstation's RAM. During these 3 seconds no serial data is being sent; therefore the buffer size on the LPM should be 2K bytes. (Note: even though only a 2K byte FIFO is required, this design used an 8 Kbyte FIFO.)

To keep the LPM's buffer full the serial data rate must be greater than 6.6Kbps. The two bit rates which the

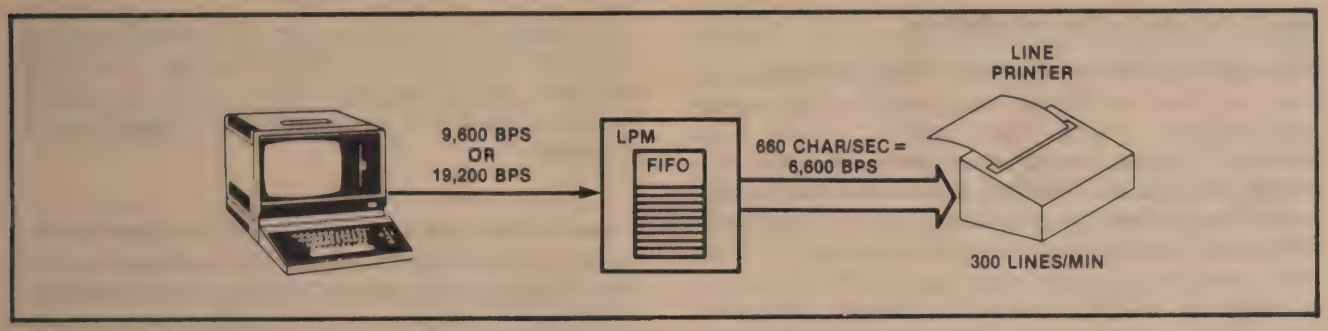


Figure 21. LPM Buffer Size Calculation

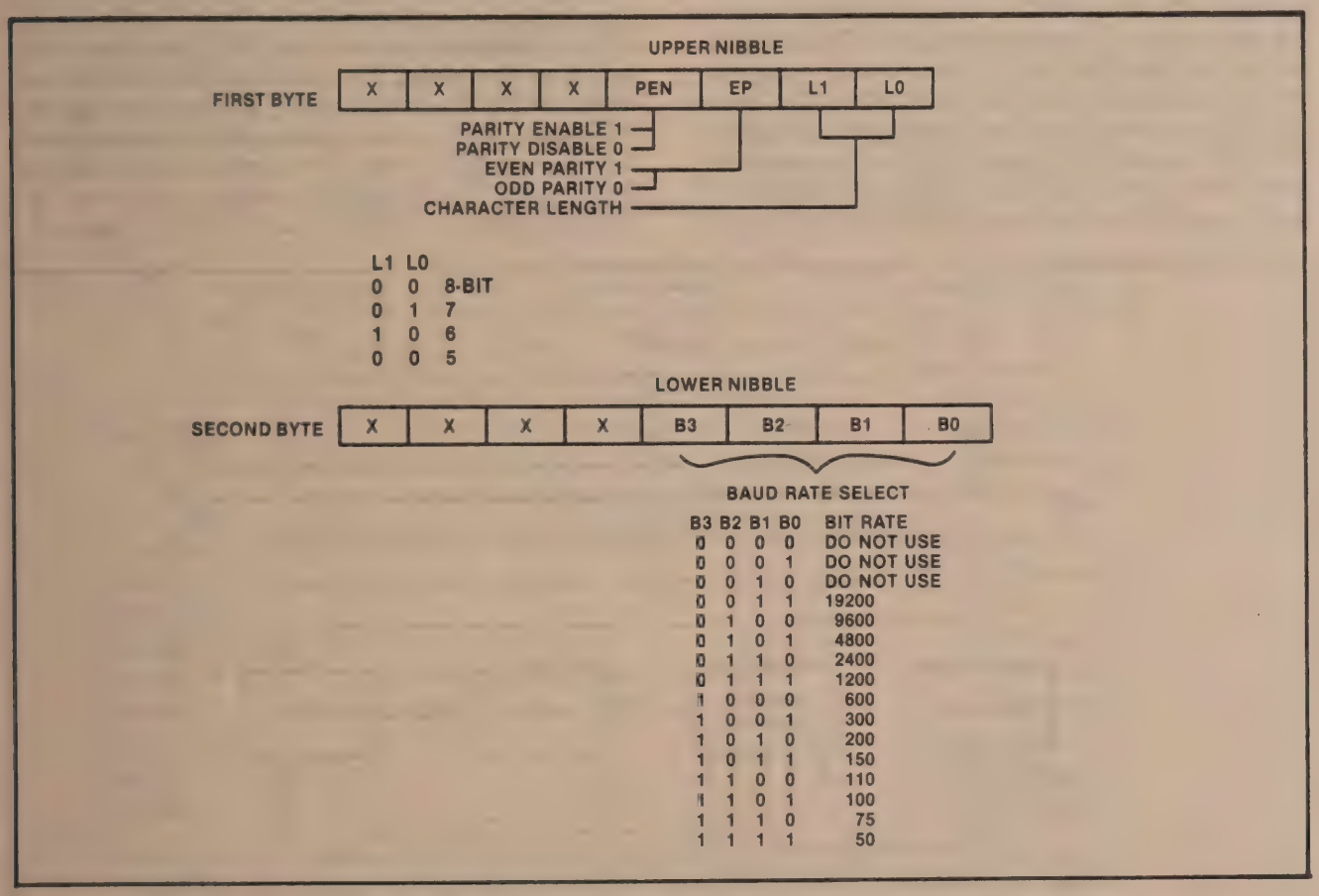


Figure 22. Programming Words Format for LPM

workstations use are 9.6Kbps and 19.2Kbps. The $\overline{CTS}$ signal is used to control the flow of the serial data so that the LPM buffer will not overflow.

Each serial port on the LPM can have a different bit rate, character length, and parity format. These parameters are programmable through the serial port. When the LPM powers up, or is reset, it expects a bit rate of 9600 bps, 7 bit characters, and odd parity. When a serial port receives an ASCII ESC character (1BH), it puts that port in the program mode. The next two bytes will program these three parameters. Only the lower nibbles of these two bytes are used, and the upper nibbles are discarded. The format of these programming words is given in Figure 22. If the word following the ESC is an ASCII NUL (0), the LPM will exit from the programming mode and not change any of its parameters.

Description of the Hardware

Figure 23 shows a block diagram of the LPM. In addition to the standard components of most microprocessor systems such as CPU, RAM, and ROM this particular design requires a UART, timers, parallel I/O and an interrupt controller. The MUART is the ideal choice for this design since it integrates these four functions onto one device.

The eight serial I/O ports use four signals: Transmit Data (TxD), Receive Data (RxD), Request To Send ($\overline{RTS}$), and Clear To Send ($\overline{CTS}$). These four signals, controlled by the MUART, are connected to one port at a time using TTL multiplexers. The TTL multiplexers are interfaced to RS-232 transceivers to be electrically compatible with the RS-232 spec. The serial port select address is derived from three bits of the MUART's parallel I/O port (Port 1). Two more bits from Port 1 control $\overline{CTS}$ and $\overline{RTS}$, and another bit lights up an LED to indicate when the LPM's buffer is full. Parallel Port 2 and two bits from Port 1 are connected to the line printer implementing a two-wire byte handshake transfer. These signals are passed through a line driver so that they can reliably drive a long cable.

There are three timing functions needed for the LPM: a scan timer, a debounce timer, and a recieve timeout. The Scan timer determines the amount of time spent sampling $\overline{RTS}$ on each port before the next port is addressed. By using one of the MUART's timers to do this function, the CPU is free to perform other functions instead of implementing the timer in software. If $\overline{RTS}$ is recognized as true, the CPU branches into a debounce procedure. This procedure uses another one of the MUART's timers to wait 10 msec then sample $\overline{RTS}$ again, thus preventing any glitches from registering as a false $\overline{RTS}$. The receive timeout timer uses two 8-bit timers in the cascaded mode to measure an 18-second interval. After a valid $\overline{RTS}$ is recognized,

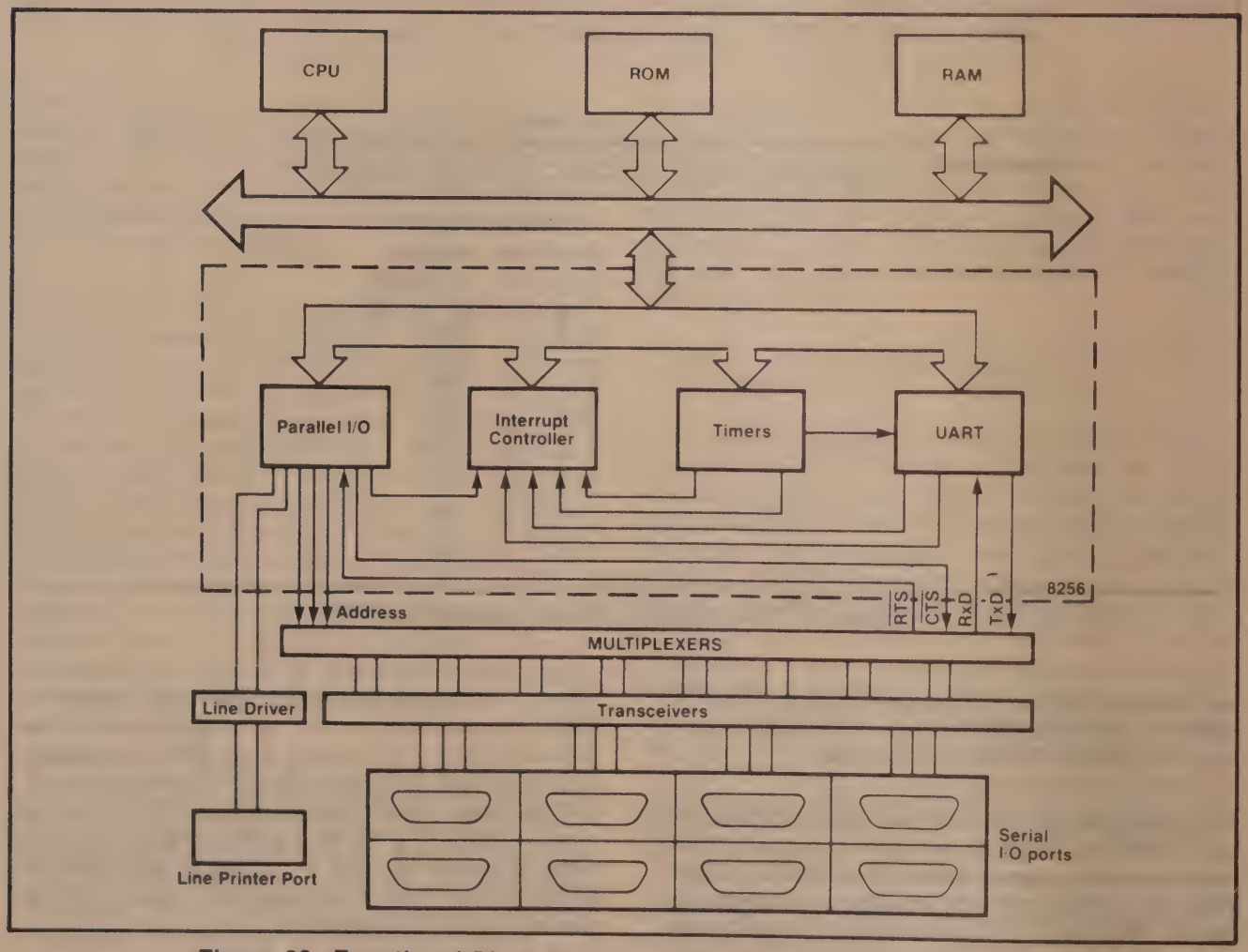


Figure 23. Functional Block Diagram of the Line Printer Multiplexer

the LPM sends back a $\overline{CTS}$ and initializes the receive timeout timer for 18 seconds. Each time a character is received by the LPM, this timer is reinitialized. If this timer times out, the LPM considers the transmission complete and returns to scanning.

The schematic diagram of the LPM is shown in Figure 24. The CPU is an 8088 used in the min mode. It is interfaced directly to the 8256. An 8282 latch is employed in the system so that nonmultiplexed bus memory can be used. A 2716 holds the entire program, and six 2016s (2K x 8 static RAMs) are used to store the buffer, temporary data, stack area, and interrupt vector table. The 2716 is located in the upper 2K of the 8088 address space (FF800-FFFFFH) so that the reset vectors can be stored starting at location FFFF0H. The RAM address space spans 0-2FFFH so that the interrupt vector table can be stored starting at location 0. The MUART is I/O mapped and its registers occupy even addresses from 0 to 1EH. Using an 8088 CPU the MUART must be placed in the 8086 mode since the $\overline{INTA}$ signal is used; hence the register addresses are all even numbers.

The line printer used provides a choice of two standard parallel interfaces: Centronics or Dataproducts. The Centronics interface uses a two-wire handshake pulsed strobe where the transmitter asserts a complete strobe pulse before an acknowledge is received. The Dataproducts interface is an interlocking two-wire handshake. The Dataproducts interface was chosen since it is directly compatible with the MUART's two-wire byte handshake. The MUART could also be connected to the Centronics interface; however, additional hardware would be necessary to generate the pulsed strobe for correct interrupt operation. Figure 25 shows the timing of the Dataproducts interface and Table 6 lists the connector pin configuration.

Table 6. Dataproducts Interface Line Functions

Signal	Description	Connector Pin
Data Request	Sent by printer to synchronize data transmission. When true, requests a character. Remains true until **Data strobe** is received, then goes false within 100 nsec.	E(return C)
Data Strobe	Sent by user system to cause printer to accept information on data lines. Should remain true until printer drops **Data Request** line. Data lines must stabilize for at least 50 nsec before **Data Strobe** is sent.	j(return m)
Data Bit 1 Data Bit 2 Data Bit 3 Data Bit 4 Data Bit 5 Data Bit 6 Data Bit 7 Data Bit 8	Bit 8 controls optional character set Refer to *Commands and Formats.*	B(return D) F(return J) L(return N) R(return T) V(return X) Z(return b) n(return k) h(return e)
VFU Control (PI)	Optional control from user system. Used for VFU control. Data Request/Strobe timing is same as for data lines.	p(return s)
Ready	Sent to user system by printer. True when no Check condition exists.	CC(return EE)
On Line	Sent to user system by printer. True when **Ready** line is true and operator has activated ON LINE Pushbutton. Enables interface activity.	y(return AA)
Interface Verify	Jumper in printer connector. Continuity informs user system that connector is properly seated.	x to v
+5V	Supply voltage for Exerciser only.	HH

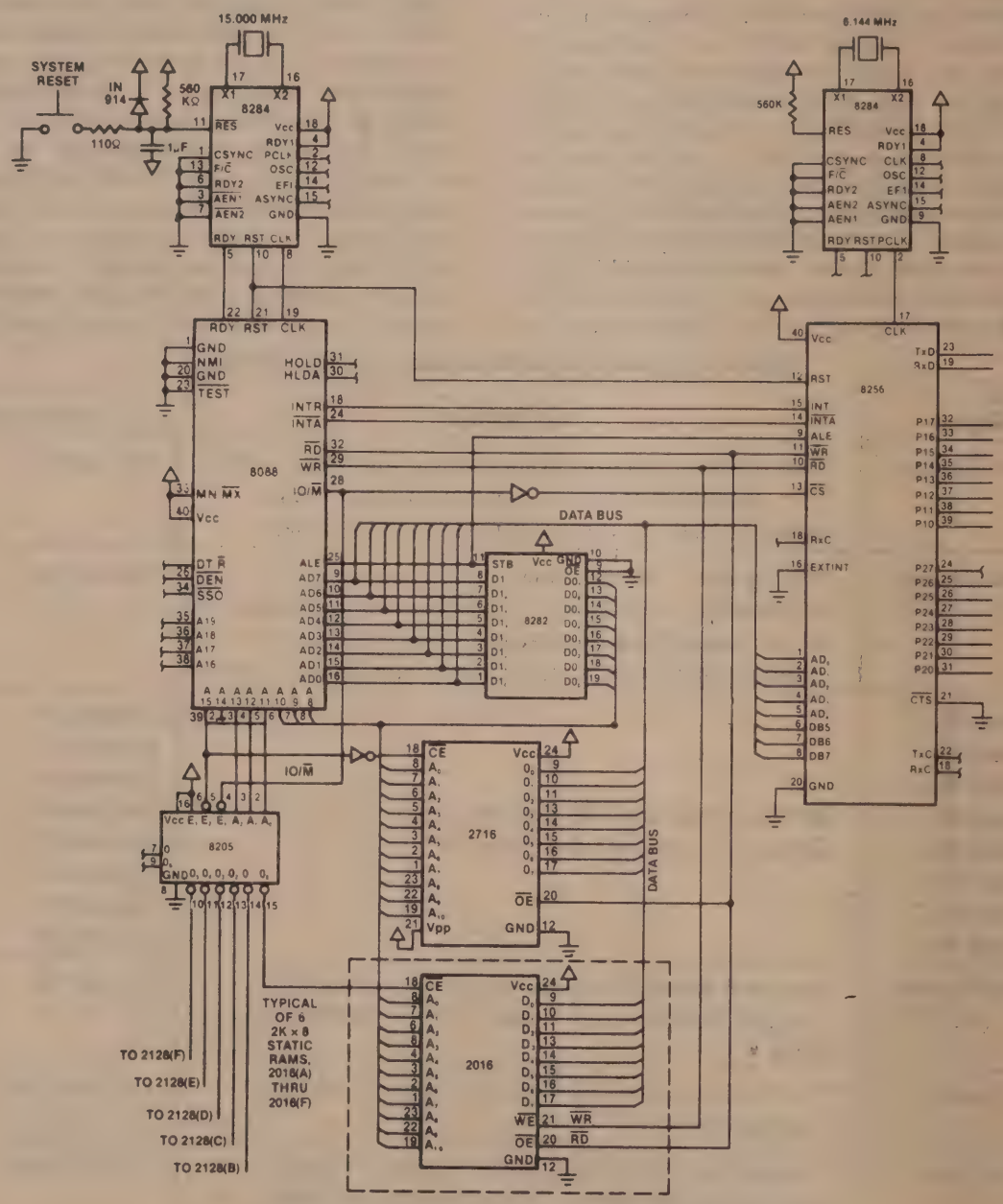


Figure 24. Schematic of LPM

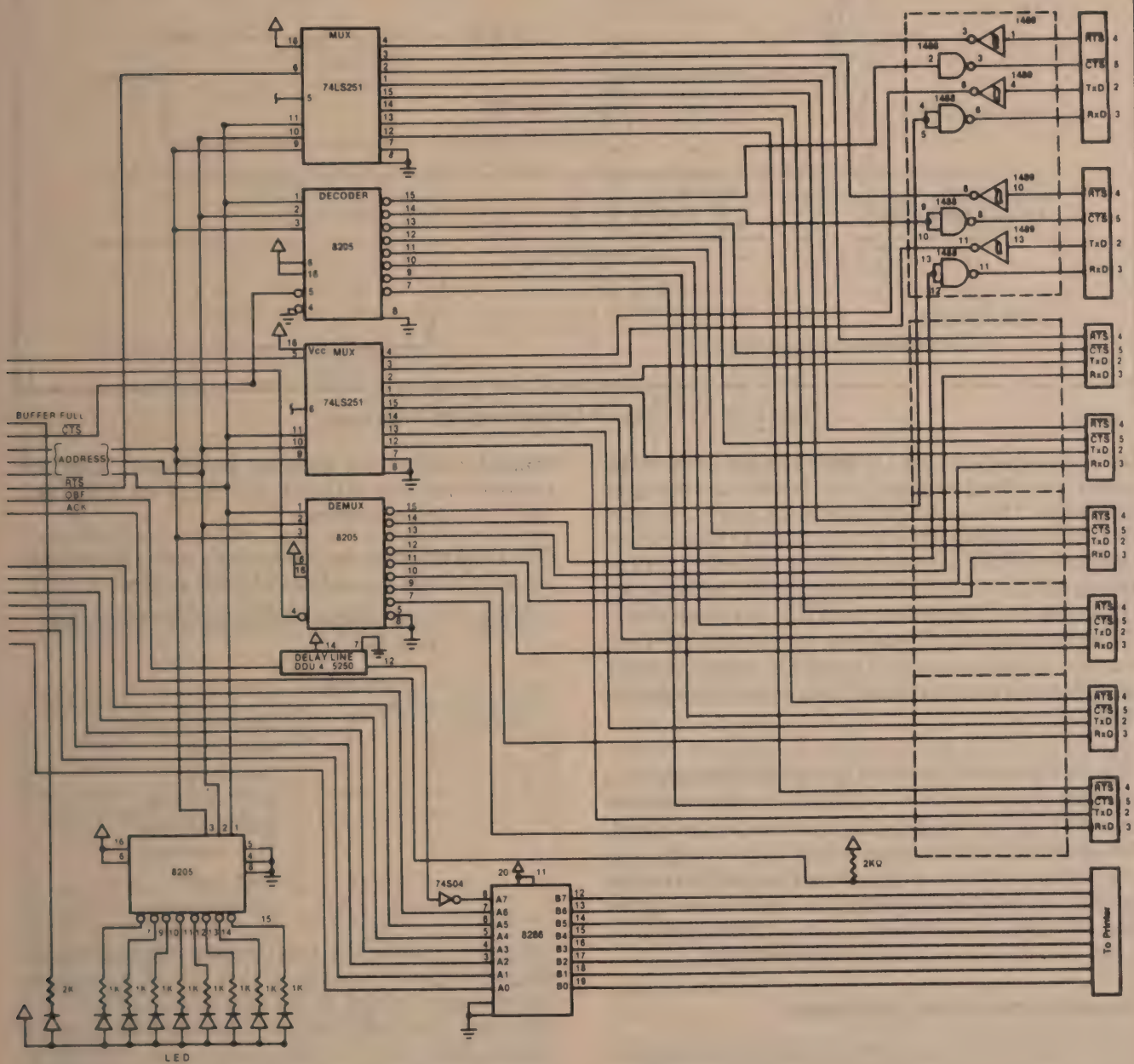


Figure 24. Schematic of LPM (Continued)

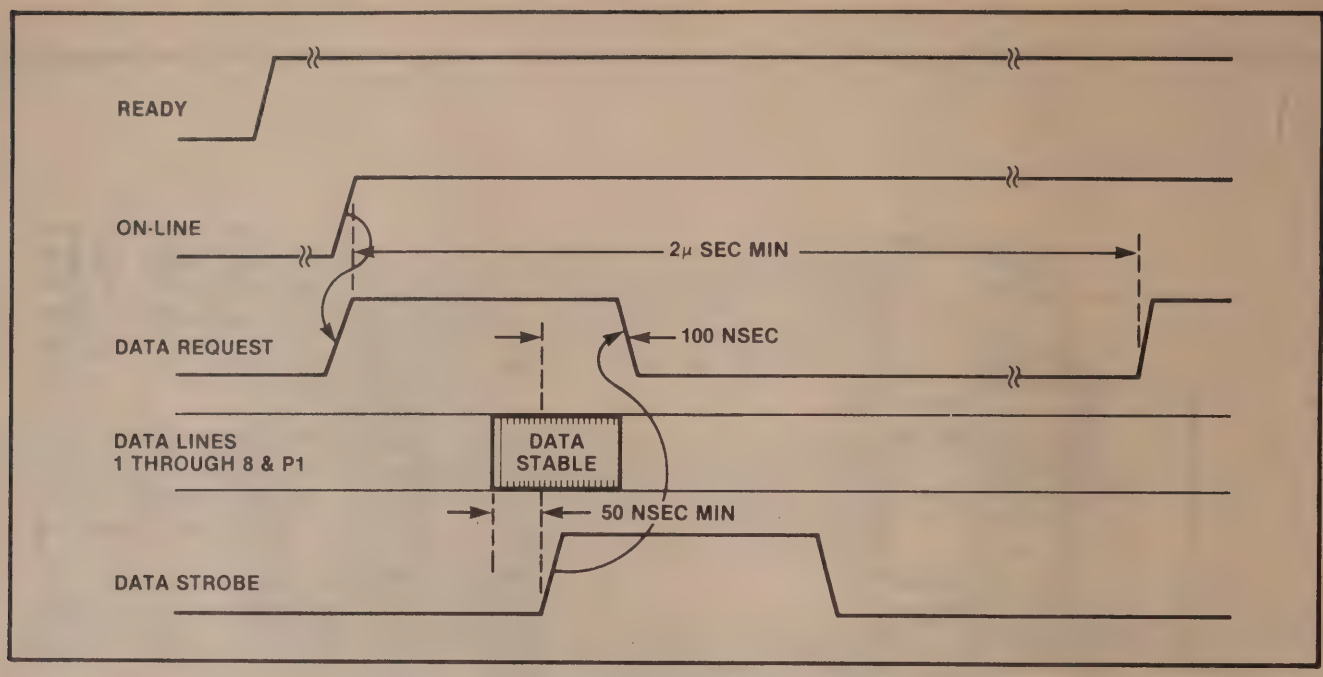


Figure 25. Timing of Dataproducts Interface

Only ten signals are used to interface the LPM to the line printer: Data Request, Data Strobe, and the eight data lines. The most significant data line is not used since the character code is 7-bit ASCII. Data Strobe connects to $\overline{OBF}$ on the MUART; however, for the Dataproducts interface this signal must be inverted. Data Request is connected to $\overline{ACK}$ on the MUART. When the line printer is ready to accept data, the Data Request signal goes high. The 8256 will not interrupt the CPU to transmit parallel data unless this signal is high.

The Dataproducts interface is slightly different from the MUART's two-wire handshake in that it latches the data on the leading edge of the strobe signal. When the MUART receives bytes it latches the data on the trailing edge. As a result the Dataproducts interface has a 50 nsec setup time for data stable to the leading edge of Data Strobe. In the LPM hardware a delay line was used to realize this setup time.

Description of the Software

The software is written in PL/M and is broken up into four separate modules, each containing several procedures. A block diagram of the software structure is given in Figure 26. The modules are identified by the dotted boxes, and the procedures are identified by the solid boxes. Two or more procedures connected by a solid line means the procedure above calls the procedure below. The procedures without any solid lines connected above are interrupt procedures. They are entered when the MUART interrupts the CPU and vectors an indirect address to it.

The LPM program uses nested interrupts; the priority of the interrupt procedures is given in Table 7.

Table 7. Line Printer Multiplexers' Interrupt Priority

	Priority	Source
Highest	0	Debounce timer
	1	Not Used
	2	Not Used
	3	Receive timer
	4	RxD Interrupt
	5	TxD Interrupt
	6	Scan timer
	7	LP Interrupt

The priority of the interrupts is not programmable but they are logically oriented so that for this application the priority is correct. In the steady state of the LPM's operation the UART will be receiving data, and the parallel port will be transmitting data. The serial receiver should be the highest priority since it can have overrun errors. This is the case because the debounce timer will be disabled, and the receive timeout interrupt will only occur when serial reception has ended. Therefore the RxD request can interrupt any other service routine, thus preventing any possibility of an overrun error.

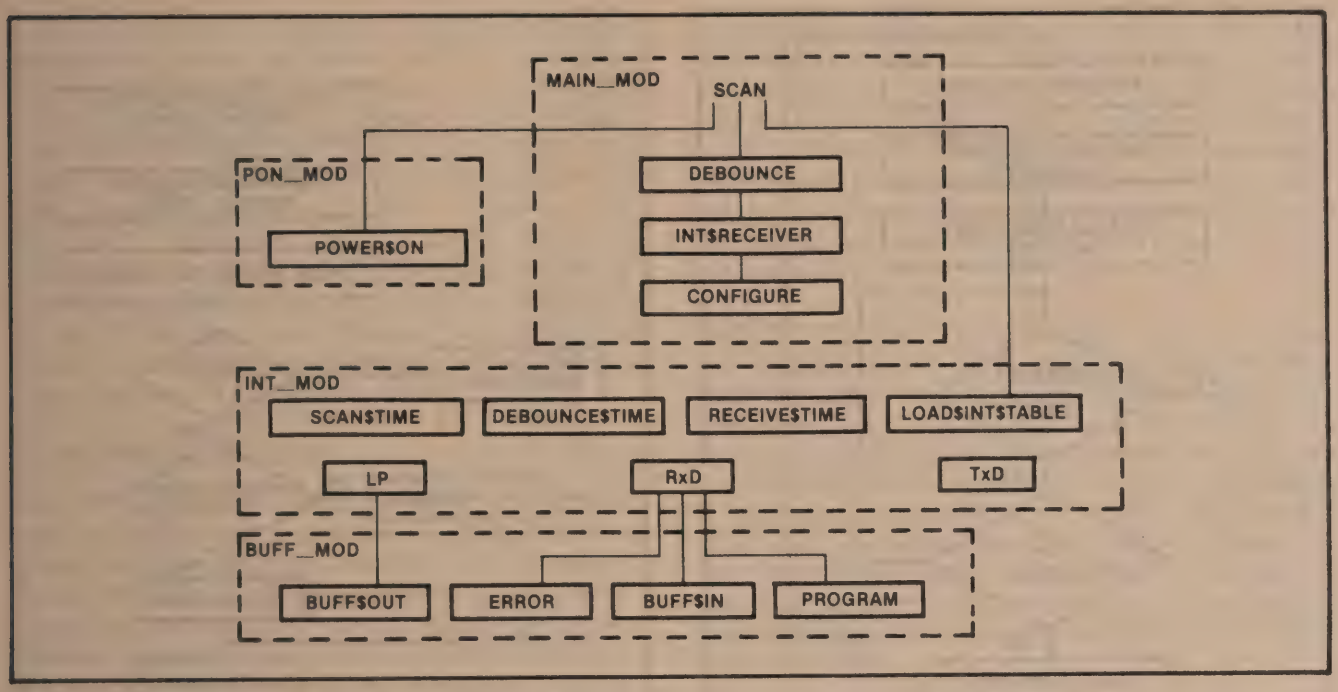


Figure 26. Block Diagram of LPM Software Structure

On power-up the CPU branches from 0FFFF0H to the INITCODE routine which is included in the machine code by the MDS locater utility. INITCODE initializes the 8088's segment registers, stack pointer, and instruction pointer, then it disabled interrupts and jumps into MAIN_MOD. The first executable instruction in MAIN_MOD calls POWERSON, which initializes the MUART, flags, variables, and arrays. The MAIN_MOD calls LOADSINTSTABLE, which initializes the interrupt vector table. The CPU's interrupt is then enabled and the program enters into a DO FOREVER loop which scans the eight serial ports for an $\overline{RTS}$.

There are three software functions which employ the MUART's timers and interrupt controller to measure time intervals: SCAN, debounce, and INITSRECEIVER. DEBOUNCE and INITSRECEIVER procedures, employ the MUART's timers and interrupt controller to measure time intervals. The CPU remains in a loop for a specific amount of time before it proceeds with the next section of code. In this loop the CPU is waiting for a global status flag to change while servicing any interrupts which may occur. When the appropriate timer interrupt occurs, the interrupt service routine will set the global flag which causes the CPU to exit the loop and proceed to the next section of code. An example can be seen from the scan flow chart in Figure 27.

The first thing the program does before entering the loop is set the flag (in this case SCANSDELAY) TRUE. The timer is initialized and the loop is entered. As long as SCANSDELAY is TRUE the CPU will continue to sample $\overline{RTS}$. If $\overline{RTS}$ remains false for more than 100 msec, the timer interrupts the CPU and the interrupt service routine sets SCANSDELAY FALSE. This causes the CPU to exit the loop and address the next port. The process is then repeated. If $\overline{RTS}$ becomes true while it is being sampled, the DEBOUNCE procedure is called.

DEBOUNCE does nothing more than wait 10 msec and sample $\overline{RTS}$ again using the same technique discussed above. If $\overline{RTS}$ is still valid INITSRECEIVER is called, otherwise the CPU returns to scan.

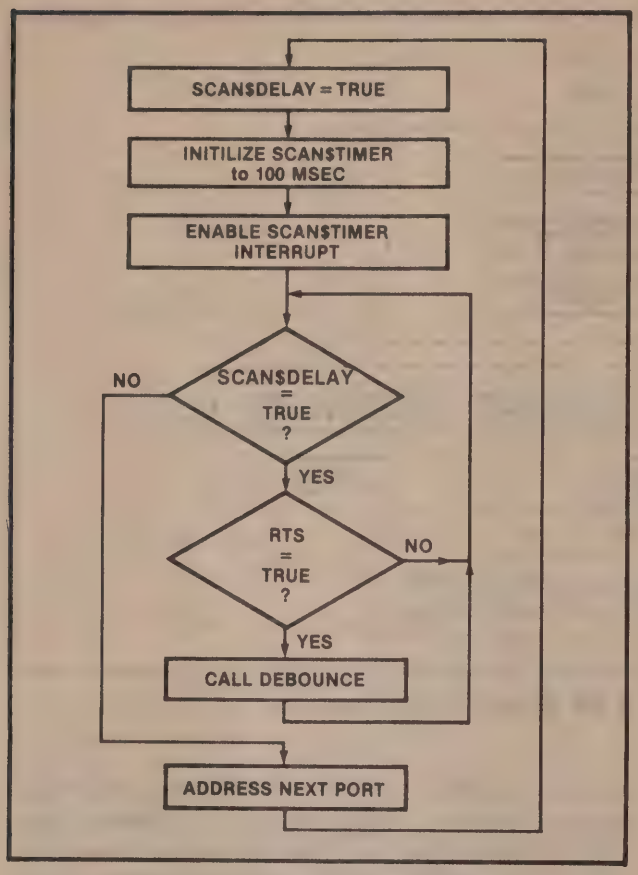


Figure 27. Scan Flow Chart

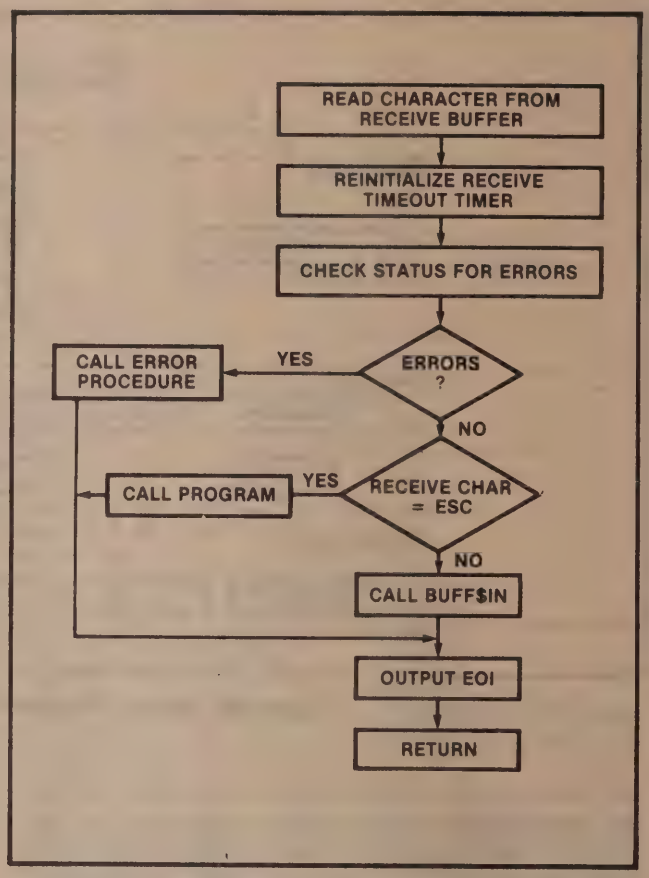


Figure 28. RxD Interrupt Procedure Flow Chart

INIT$RECEIVER calls CONFIGURE which programs the MUART for the bit rate, number of bits in a character, and parity format. This information is stored in an array called SERIAL$FORMAT, which contains a byte for each port. The bytes in the SERIAL$FORMAT array have the same bit definition as the two nibbles in the programming words in Figure 22. Upon returning to INIT$RECEIVER the receiver is enabled, the receive timeout timer is initialized, and the timer and receiver interrupts are enabled. $\overline{\text{CTS}}$ on the serial port is then set true, and the CPU enters a loop which does nothing except wait for 18 seconds. If no characters are received within 18 seconds, the receive timeout interrupt occurs and the loop flag is set false, which causes the CPU to exit the loop. If a character is received, a receive interrupt occurs, and the CPU vectors into the RxD interrupt service routine.

Figure 28 shows a flow chart of the RxD interrupt service routine. This routine begins by reading the receive buffer and reinitializing the receive timeout timer. There are two conditions to check for before the character can be inserted into the FIFO. First, if there are any errors in the received character, an ERROR procedure is called which reports back to the serial port what the error condition was. The character in error is discarded and the routine returns. The other condition is that if the received character is an ASCII ESC, the PROGRAM procedure is called. If neither one of these conditions occurs, the character is placed in the FIFO by the BUFF$IN procedure.

The LP interrupt routine is entered when the byte handshake interrupt request is acknowledged. This routine simply calls the BUFF$OUT procedure, which extracts a byte out of the FIFO. BUFF$OUT returns the byte to the LP interrupt procedure, which then writes it to Port 2. One small problem with getting the handshake interrupt going is that the first byte has to be written to Port 2 before the first handshake interrupt will occur. The problem is that the line printer may not be ready for the first byte. This would be indicated by DATA REQUEST being low. If the byte was written to the LP while DATA REQUEST is low, it would be lost. Note that if the handshake interrupt is enabled while DATA REQUEST is low, then DATA REQUEST goes high, the interrupt will occur without

writing the first byte. There are several ways to solve this problem. Port 1 can be read to find out what the state of the DATA REQUEST line is. If DATA REQUEST is low, the CPU can simply wait for the interrupt without writing the first byte. If DATA REQUEST is high, then the first data byte may be written. Another solution would be to write a NUL character as the first byte to Port 2. If DATA REQUEST is low, then a worthless character is lost. If DATA REQUEST is high, the NUL character would be sent to the line printer; however, it is not printed since NUL is a nonprintable character. The LPM program uses the NUL character solution.

BUFFER MANAGEMENT

The FIFO implementation uses an 8K byte array to store the characters. There are two pointers used as indexes in the array to address the characters: IN$POINTER and OUT$POINTER. IN$POINTER points to the location in the array which will store the next byte of data inserted. OUT$POINTER points to the next byte of data which will be removed from the array. Both IN$POINTER and OUT$POINTER are declared as words. Figure 29 illustrates the FIFO in a block diagram.

The BUFF$IN procedure receives a byte from the RxD interrupt routine and stores it in the array location pointed to by IN$POINTER, then IN$POINTER is incremented. Similarly, when BUFF$OUT is called

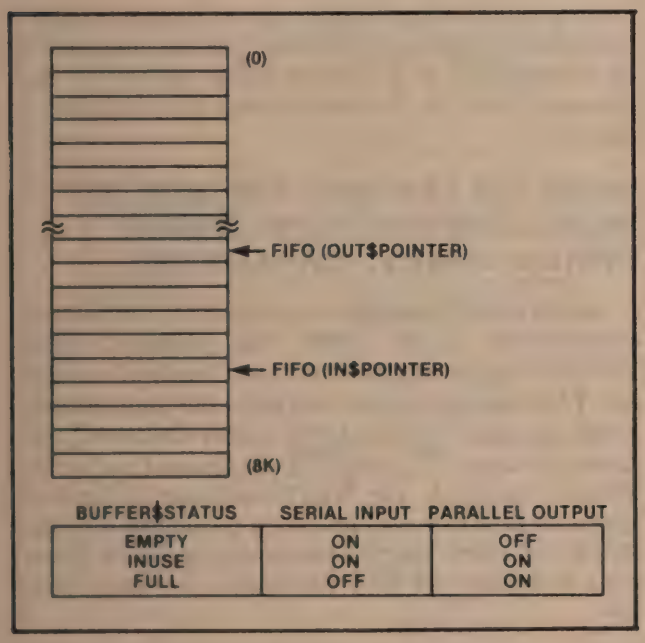


Figure 29. FIFO Structure and Status

by the LP interrupt routine, the byte in the array pointed to by OUT$POINTER is read. OUT$POINTER is incremented, and the byte which was read is passed back to the LP interrupt routine. Since IN$POINTER and OUT$POINTER are always incremented, they must be able to roll over when they hit the top of the 8K byte address space. This is done by clearing the upper three bits of each pointer after it is incremented.

IN$POINTER and OUT$PONTER not only point to the locations in the FIFO, they also indicate how many bytes are in the FIFO and whether the FIFO is full or empty. When a character is placed into the FIFO and IN$POINTER is incremented, the FIFO is full if IN$POINTER equals OUT$POINTER. When a character is read from the FIFO and OUT$POINTER is incremented, the FIFO is empty if OUT$POINTER equals IN$POINTER. If the buffer is neither full nor empty, then it is in use. A byte called BUFFER$STATUS is used to indicate one of these three conditions.

The software uses the buffer status information to control the flow into and out of the FIFO. When the FIFO is empty the handshake interrupt must be turned off. When the FIFO is full, $\overline{CTS}$ must be sent false so that no more data will be received. If the buffer status is in use, $\overline{CTS}$ is true and the handshake interrupt is enabled.

Figure 30 shows the flow chart of the BUFF$IN procedure. The BUFF$IN procedure begins by checking the BUFFER$STATUS. If it is empty and the character to be inserted into the FIFO is a CR or LF, the handshake interrupt is enabled, a NUL character is output, and the BUFFER$STATUS is set to INUSE. The character passed to BUFF$IN from RxD is put into the FIFO. If the FIFO is now full, the BUFFER$STATUS is set to FULL, $\overline{CTS}$ is set false, and the buffer full LED is turned on.

Figure 31 shows the flow chart of the BUFF$OUT procedure. After the character is read from the FIFO, the FIFO is tested to determine if it is empty. If it is not empty, the BUFFER$STATUS is FULL and there are 200 bytes available in the FIFO, serial data reception is reenabled, and the FIFO fills again. While data is being received from the workstation, $\overline{CTS}$ toggles high and low, filling up and emptying the last 200 bytes in the FIFO. Referring to the top of the flow chart (FIFO empty test) if it's empty, the BUFFER$STATUS is set to EMPTY, and the handshake interrupt is disabled. During this time all interrupts

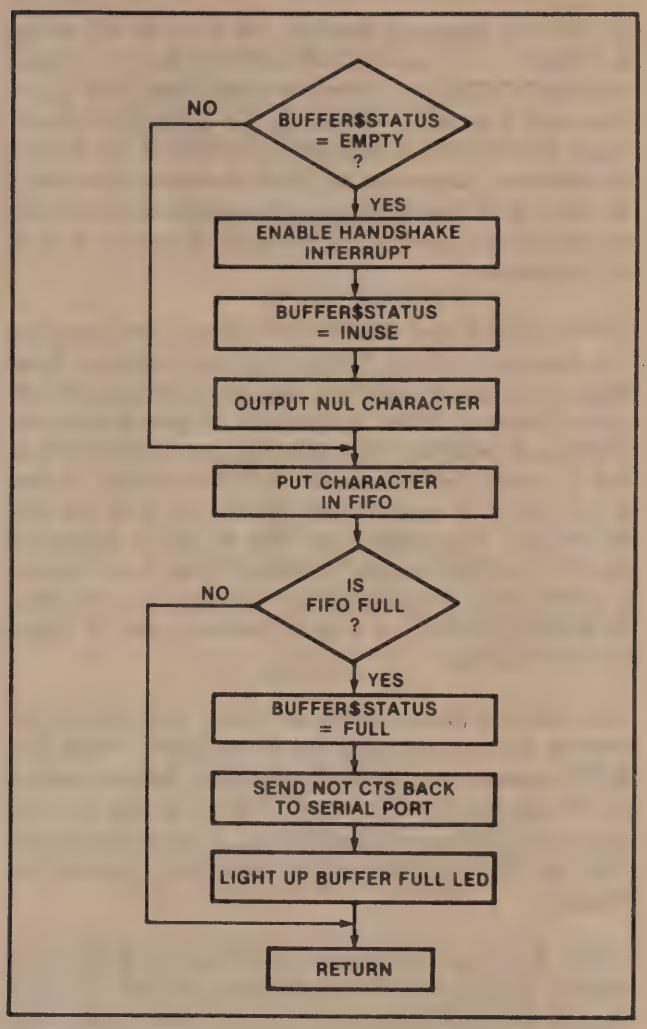


Figure 30. Flow Chart of the BUFF$IN Procedure

are disabled at the CPU. (Remember that the RxD interrupt routine can interrupt the LP and BUFF$OUT procedures since it has a higher priority, and the MUART is in the nested mode.)

If the CPU interrupt was not disabled during this time, the following events could occur which would cause the LPM to crash. Assume that the RxD interrupt occured where the asterisk is in the flow chart, after BUFFER$STATUS is set to EMPTY. The BUFF$IN procedure would set BUFFER$STATUS to INUSE and enable the handshake interrupt. When the RxD interrupt routine returned to BUFF$OUT, the handshake interrupt is disabled, but the BUFFER$STATUS is INUSE. The handshake interrupt could never be reenabled, and the FIFO would fill up. This is known as a critical section of code. Suspicion should arise for a critical section of code when two or more nested interrupt routines can affect the same status. One solution is to disable the interrupt flag at the CPU while the status and conditional operations are being modified.

The flow chart for the TxD interrupt procedure is given in Figure 32. For this program five different messages can be transmitted, and they are stored in ROM. It is possible to download the messages into a dedicated RAM buffer; however, the RAM buffer would have to be as large as the largest message. A more efficient way to transmit the messages is to read them from ROM. In this case the address of the first byte of the message would have to be accessible by the transmit interrupt procedure. Since parameters cannot be passed to interrupt procedures, this message pointer is declared PUBLIC in one module and EXTERNAL in the other modules.

To get the transmit interrupt started, the first byte of the message must be written to the transmit buffer. When a section of code decides to transmit a message serially, it loads the global message pointer with the address of the first byte of the message, enables the transmit interrupt, and calls the TxD interrupt procedure. Calling the TxD interrupt procedure writes the first byte to the transmit buffer to initiate transmit interrupts. This can be done by calling PL/M's built-in procedure CAUSE$INTERRUPT.

The transmit interrupt routine checks each byte before it writes it to the transmit buffer. The last character in each message is a 0, so if the character fetched is 0, the transmit interrupt is disabled and the character is ignored.

USING THE LPM WITH THE INTELLEC® MICROCOMPUTER DEVELOPMENT SYSTEM, SERIES II OR SERIES III

A special driver program was written for the MDS to communicate to the LPM. This program, called WRITE, reads a specified file from the disk, expands any TAB characters, and transmits the data through Serial Channel 2 to the LPM. Serial Channel 2 was chosen because $\overline{CTS}$ and $\overline{RTS}$ are brought out to the RS-232 connector. The WRITE program is listed in appendix B. It was also necessary to modify the boot ROM of the development system so that Serial Channel 2 initializes with $\overline{RTS}$ false and a bit rate of 9600 bps.

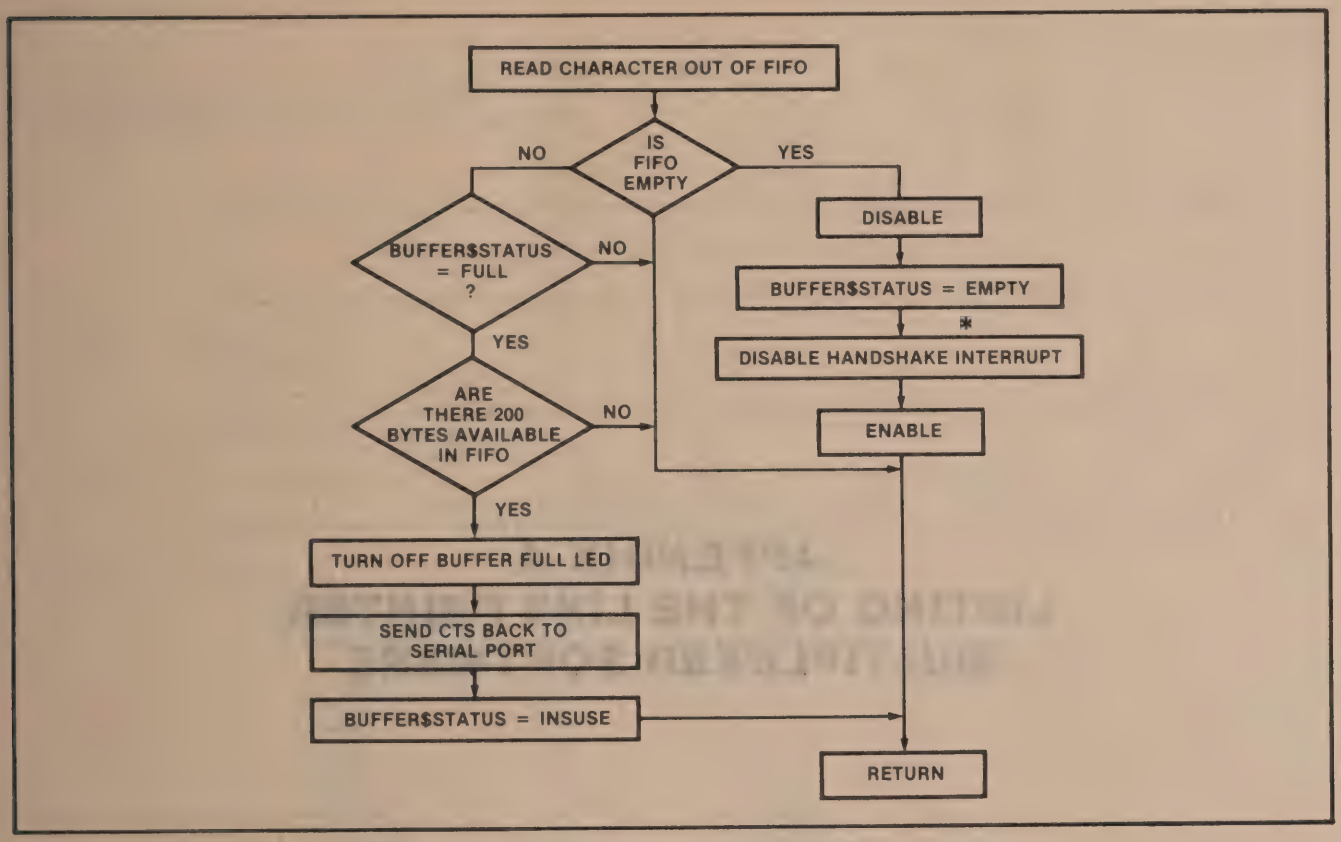


Figure 31. Flow Chart of the BUFF$OUT Procedure

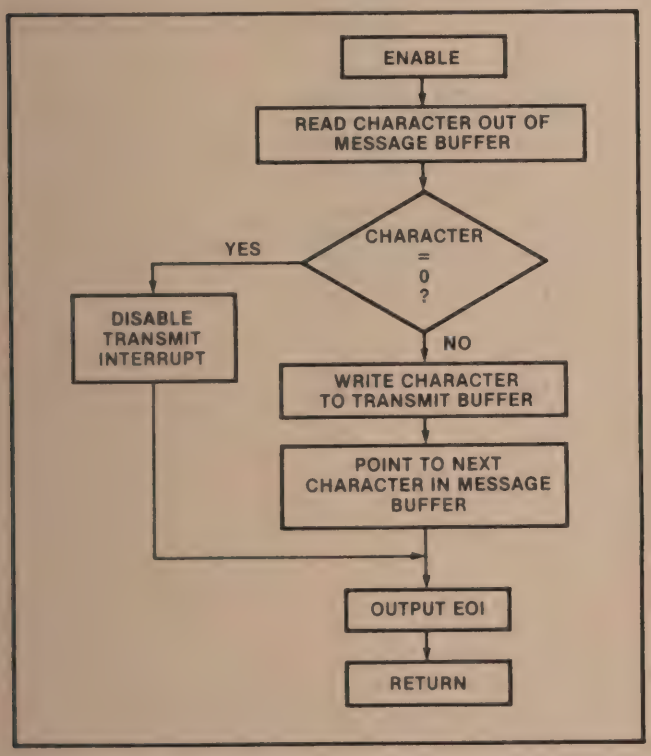


Figure 32. Flow Chart for TxD Interrupt Procedure

APPENDIX A
LISTING OF THE LINE PRINTER MULTIPLEXER SOFTWARE

```
PL/M-86 COMPILER    MAINMOD

SERIES-III PL/M-86 V1.0 COMPILATION OF MODULE MAINMOD
OBJECT MODULE PLACED IN :F1:MAIN.OBJ
COMPILER INVOKED BY:  PLM86.86 :F1:MAIN.SRC

          /*************************************************************************
           *                                                                       *
           *            MAIN MODULE FOR THE LINE PRINTER MULTIPLEXER               *
           *                                                                       *
           *************************************************************************/

          $DEBUG
  1       MAIN$MOD:DO;

          /*************************************************************************
          * PORT 1 BIT CONFIGURATION                                               *
          *                                                                        *
          *  BUFFER FULL   CTS      ADDRESS    RTS   TWO WIRE HANDSHAKE            *
          *       B7        B6     B5 B4 B3    B2        B1 B0                     *
          *************************************************************************/

  2   1   DECLARE LIT              LITERALLY        'LITERALLY',
                  TRUE             LIT              '0FFH',
                  FALSE            LIT              '0',
                  FOREVER          LIT              'WHILE 1',

                  CMD$1            LIT              '0',     /*8256 REGISTERS*/
                  CMD$2            LIT              '2',
                  CMD$3            LIT              '4',
                  MODE             LIT              '6',
                  PORT$1$CTRL      LIT              '8',
                  SET$INT          LIT              '0AH',
                  INT$EN           LIT              '0AH',
                  RST$INT          LIT              '0CH',
                  INT$ADDR         LIT              '0CH',
                  TX$BUFF          LIT              '0EH',
                  RX$BUFF          LIT              '0EH',
                  PORT$1           LIT              '10H',
                  PORT$2           LIT              '12H',
                  DEBOUNCE$TIMER   LIT              '14H',
                  SCAN$TIMER       LIT              '1AH',
                  RECEIVE$TIMER    LIT              '1CH',
                  STATUS$REG       LIT              '1EH',

                  SCAN$INT         LIT              '40H',
                  DEBOUNCE$INT     LIT              '01H',
                  RECEIVER$INT     LIT              '10H',
                  TIME$OUT$INT     LIT              '08H',
                  TRANSMIT$INT     LIT              '20H',

                  EMPTY            LIT              '0',
                  INUSE            LIT              '1',
                  FULL             LIT              '2',

                  RTS              LIT              '(INPUT(PORT$1) AND 04H)',
```

```
PL/M-86 COMPILER    MAINMOD

                    BEGIN           LABEL           PUBLIC,

                    TEMP            BYTE,
                    SCAN$DELAY      BYTE            PUBLIC,
                    DEBOUNCE$DELAY  BYTE            PUBLIC,
                    RECEIVE$DELAY   BYTE            PUBLIC,
                    PORT$PTR        BYTE            PUBLIC,
                    SERIAL$FORMAT(8)BYTE            PUBLIC, /* PEN EP L1 L0 B3 B2 B1 B0 */

                    MESSAGE$PTR     POINTER         EXTERNAL,
                    J               BYTE            EXTERNAL,
                    OK(1)           BYTE            EXTERNAL,
                    BUFFER$STATUS   BYTE            EXTERNAL;

              /*****************************************************************
               *                EXTERNAL PROCEDURE DECLARATIONS                *
               *****************************************************************/

  3   1       POWER$ON: PROCEDURE EXTERNAL;
  4   2       END POWER$ON;

  5   1       LOAD$INT$TABLE: PROCEDURE EXTERNAL;
  6   2       END LOAD$INT$TABLE;

              /*****************************************************************
               *      SET THE BIT RATE AND DATA FORMAT FOR THE SERIAL PORT      *
               *****************************************************************/

  7   1       CONFIGURE: PROCEDURE ; /*Initialize bit rate and data format*/
  8   2       TEMP=SERIAL$FORMAT(SHR(PORT$PTR,3));
  9   2       OUTPUT(CMD$1)=((SHL(TEMP,2) AND 0C0H) OR 03H);
 10   2       OUTPUT(CMD$2)=(TEMP OR 30H);
 11   2       END CONFIGURE;

              /*****************************************************************
               *                INITIALIZE SERIAL RECEIVER                     *
               *****************************************************************/

 12   1       INIT$RECEIVER   PROCEDURE;
 13   2       CALL CONFIGURE;                    /*Initialize 8256 serial port*/
 14   2       RECEIVE$DELAY=TRUE;
 15   2       OUTPUT(CMD$3)=0C0H;                /*Enable serial receiver*/
 16   2       OUTPUT(RECEIVE$TIMER)=70;          /*18 second TIME$OUT*/
 17   2       OUTPUT(SET$INT)=18H;               /*Enable RECEIVER and TIME$OUT interrupts*/
 18   2       IF (BUFFER$STATUS<>FULL)
                  THEN
 19   2           OUTPUT(PORT$1)=(INPUT(PORT$1) AND 0BFH); /*Send CTS TRUE*/

 20   2           DO WHILE RECEIVE$DELAY=TRUE; /* Wait here while receiving serial data */
 21   3           END;

                      /* After 18 seconds of not receiving a character, proceed */

 22   2       OUTPUT(SET$INT)=TRANSMIT$INT;      /* Send the terminating message */
 23   2       J=0;
 24   2       MESSAGE$PTR= @OK(0);
 25   2       CAUSE$INTERRUPT (45H);
```

```
PL/M-86 COMPILER    MAINMOD


 26   2      OUTPUT(PORT$1)=(INPUT(PORT$1) OR 40H); /*Send CTS FALSE*/
 27   2      OUTPUT(RST$INT)=18H;              /*Clear RECEIVER and TIMER Interrupts*/
 28   2      OUTPUT(CMD$3)=40H;                /*Disable serial receiver*/
 29   2      END INIT$RECEIVER;

             /*****************************************************************
              *                          DEBOUNCE RTS                         *
              *****************************************************************/

 30   1      DEBOUNCE: PROCEDURE;
 31   2      DEBOUNCE$DELAY=TRUE;
 32   2      OUTPUT(DEBOUNCE$TIMER)=10; /* 10 msec debounce time delay */
 33   2      OUTPUT(SET$INT)=DEBOUNCE$INT;
 34   2          DO WHILE DEBOUNCE$DELAY=TRUE;
 35   3          END;
 36   2      IF RTS=0 THEN CALL INIT$RECEIVER;
 38   2      END DEBOUNCE;

             /*****************************************************************
              *                        BEGIN MAIN PROGRAM                     *
              *****************************************************************/


 39   1      BEGIN: CALL POWER$ON;

 40   1      CALL LOAD$INT$TABLE;

 41   1      ENABLE;

 42   1      DO FOREVER;

 43   2      SCAN$DELAY=TRUE;
 44   2      OUTPUT(SCAN$TIMER)=100;    /*Spend 100 msec on each serial port sampling RTS*/
 45   2      OUTPUT(SET$INT)=SCAN$INT;

 46   2          DO WHILE SCAN$DELAY=TRUE;    /*Sample RTS*/
 47   3            IF RTS=0
                     THEN
 48   3                CALL DEBOUNCE;
 49   3          END;


 50   2      TEMP=INPUT(PORT$1);               /*Increment PORT$PTR*/
 51   2      PORT$PTR=TEMP AND 38H;
 52   2      TEMP=TEMP AND (NOT 38H);
 53   2      PORT$PTR=(PORT$PTR+8) AND 38H;

 54   2      OUTPUT(PORT$1)=TEMP OR PORT$PTR; /*Look at next serial port*/
 55   2      END; /*DO FOREVER*/
 56   1      END MAIN$MOD;


MODULE INFORMATION:

    CODE AREA SIZE     = 011CH    284D


PL/M-86 COMPILER    MAINMOD


    CONSTANT AREA SIZE = 0000H      0D
    VARIABLE AREA SIZE = 000DH     13D
    MAXIMUM STACK SIZE = 000CH     12D
    159 LINES READ
    0 PROGRAM WARNINGS
    0 PROGRAM ERRORS

END OF PL/M-86 COMPILATION
```

```
PL/M-86 COMPILER    INTMOD

SERIES-III PL/M-86 V1.0 COMPILATION OF MODULE INTMOD
OBJECT MODULE PLACED IN .F1.INT OBJ
COMPILER INVOKED BY:  PLM86.86  F1.INT SRC

                /*********************************************************************
                 *                                                                   *
                 *        INTERRUPT MODULE: CONTAINS ALL INTERRUPT ROUTINES          *
                 *                          PLUS LOAD INTERRUPT TABLE PROCEDURE      *
                 *                                                                   *
                 *********************************************************************/

                $DEBUG
  1             INT$MOD:DO;
                $NOLIST

  3   1         DECLARE
                        ESC              LIT              '1BH',
                        SCAN$DELAY       BYTE             EXTERNAL,
                        DEBOUNCE$DELAY   BYTE             EXTERNAL,
                        RECEIVE$DELAY    BYTE             EXTERNAL,
                        MESSAGE$PTR      POINTER          EXTERNAL,
                        J                BYTE             EXTERNAL,

                /*********************************************************************
                 *                MESSAGES SENT TO SERIAL PORTS                      *
                 *********************************************************************/

                        OK (*) BYTE PUBLIC DATA ('TRANSMISSION COMPLETE',0AH,0DH,00),
                        BREAK (*) BYTE PUBLIC DATA ('BREAK DETECT ERROR',0AH,0DH,00),
                        PARITY (*)BYTE PUBLIC DATA ('PARITY ERROR DETECTED',0AH,0DH,00),
                        FRAME (*) BYTE PUBLIC DATA ('FRAMING ERROR DETECTED',0AH,0DH,00),
                        OVER$RUN(*)BYTE PUBLIC DATA('OVER RUN ERROR DETECTED',0AH,0DH,00);

                /*********************************************************************
                 *        EXTERNAL PROCEDURES CALLED BY THE INTERRUPT ROUTINES       *
                 *********************************************************************/
  4   1         ERROR:PROCEDURE (STATUS) EXTERNAL;
  5   2         DECLARE STATUS  BYTE;
  6   2         END ERROR;

  7   1         PROGRAM.PROCEDURE EXTERNAL;
  8   2         END PROGRAM;

  9   1         BUFF$IN:PROCEDURE (CHAR) EXTERNAL;
 10   2         DECLARE CHAR    BYTE;
 11   2         END BUFF$IN;

 12   1         BUFF$OUT.PROCEDURE BYTE EXTERNAL;
 13   2         END BUFF$OUT;

                /*********************************************************************
                 *                    LOAD THE INTERRUPT TABLE                       *
                 *********************************************************************/

 14   1         LOAD$INT$TABLE:PROCEDURE PUBLIC;
```

```
 15   2      CALL SET$INTERRUPT (40H,DEBOUNCE$TIME);
 16   2      CALL SET$INTERRUPT (43H,RECEIVE$TIME);
 17   2      CALL SET$INTERRUPT (44H,RXD);
 18   2      CALL SET$INTERRUPT (45H,TXD);
 19   2      CALL SET$INTERRUPT (46H,SCAN$TIME);
 20   2      CALL SET$INTERRUPT (47H,LP);

 21   2      END LOAD$INT$TABLE;

             /*********************************************************************
              *                    INTERRUPT ROUTINES                             *
              *********************************************************************/


             /*********************************************************************
              *                 SET SCAN DELAY FLAG FALSE                         *
              *********************************************************************/

 22   1      SCAN$TIME:PROCEDURE INTERRUPT 46H;

 23   2      ENABLE;
 24   2      SCAN$DELAY=FALSE;
 25   2      OUTPUT(CMD$3)=88H;                    /*Output end for nested mode*/
 26   2      END SCAN$TIME;

             /*********************************************************************
              *                 SET DEBOUNCE DELAY FLAG FALSE                     *
              *********************************************************************/

 27   1      DEBOUNCE$TIME:PROCEDURE INTERRUPT 40H;
 28   2      DEBOUNCE$DELAY=FALSE;
 29   2      OUTPUT(CMD$3)=88H;
 30   2      END DEBOUNCE$TIME;

             /*********************************************************************
              *                 SET RECEIVE DELAY FLAG FALSE                      *
              *********************************************************************/

 31   1      RECEIVE$TIME:PROCEDURE INTERRUPT 43H;
 32   2      ENABLE;
 33   2      RECEIVE$DELAY=FALSE;
 34   2      OUTPUT(CMD$3)=88H;
 35   2      END RECEIVE$TIME;

             /*********************************************************************
              *                  READ SERIAL RECEIVE BUFFER                       *
              *********************************************************************/

 36   1      RXD:PROCEDURE INTERRUPT 44H;

 37   2      DECLARE
                     STATUS          BYTE,
                     CHAR            BYTE;
 38   2      CHAR=INPUT(RX$BUFF);
 39   2      OUTPUT(RECEIVE$TIMER)=70,          /* REINITIALIZE RECEIVE TIME OUT */
 40   2      STATUS=INPUT(STATUS$REG) AND 0FH;
```

```
PL/M-86 COMPILER     INTMOD

 41   2          IF STATUS<>0
                     THEN
 42   2                CALL ERROR (STATUS);

 43   2          ELSE IF CHAR=ESC
                         THEN
 44   2                    CALL PROGRAM;
                ELSE
 45   2              CALL BUFF$IN (CHAR);
 46   2      OUTPUT(CMD$3)=88H;
 47   2      END RXD;

             /*****************************************************************
              *                  SEND A BYTE TO THE LINE PRINTER              *
              *****************************************************************/

 48   1      LP:PROCEDURE INTERRUPT 47H;
 49   2      ENABLE;
 50   2      OUTPUT(PORT$2)=BUFF$OUT;
 51   2      OUTPUT(CMD$3)=88H;
 52   2      END LP;

             /*****************************************************************
              *                  SEND A BYTE TO THE SERIAL PORTS              *
              *****************************************************************/

 53   1      TXD:PROCEDURE INTERRUPT 45H;
 54   2      DECLARE
                     MESSAGE BASED MESSAGE$PTR (1) BYTE,
                     I        BYTE;
 55   2      ENABLE;
 56   2      I=MESSAGE(J),
 57   2          IF I<>0
                    THEN OUTPUT(TX$BUFF)=I;
 59   2          ELSE OUTPUT(RST$INT)=TRANSMIT$INT;
 60   2          J=J+1;
 61   2          OUTPUT(CMD$3)=88H;
 62   2      END TXD;

 63   1      END INT$MOD;


MODULE INFORMATION:

     CODE AREA SIZE     = 01BDH    445D
     CONSTANT AREA SIZE = 0078H    120D
     VARIABLE AREA SIZE = 0003H      3D
     MAXIMUM STACK SIZE = 0022H     34D
     181 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS

END OF PL/M-86 COMPILATION
```

```
PL/M-86 COMPILER    BUFFMOD

SERIES-III PL/M-86 V1.0 COMPILATION OF MODULE BUFFMOD
OBJECT MODULE PLACED IN :F1:BUFF.OBJ
COMPILER INVOKED BY:  PLM86.86 :F1:BUFF.SRC


            /*******************************************************************
             *                                                                 *
             *    BUFFER MODULE:  INSERTS AND REMOVES CHARACTERS FROM FIFO     *
             *                    REPORTS SERIAL RECEIVE ERRORS AND            *
             *                    RE-PROGRAMS SERIAL PORTS                     *
             *                                                                 *
             *******************************************************************/

            $DEBUG
  1         BUFF$MOD: DO;
            $NOLIST

  3   1     DECLARE
                    MESSAGE$PTR      POINTER        PUBLIC,
                    J                BYTE           PUBLIC,
                    OK(1)            BYTE           EXTERNAL,
                    BREAK(1)         BYTE           EXTERNAL,
                    PARITY(1)        BYTE           EXTERNAL,
                    FRAME(1)         BYTE           EXTERNAL,
                    OVER$RUN(1)      BYTE           EXTERNAL,
                    SERIAL$FORMAT(1)BYTE            EXTERNAL,
                    PORT$PTR         BYTE           EXTERNAL,

                    FIFO(8192)       BYTE,
                    IN$POINTER       WORD           PUBLIC,
                    OUT$POINTER      WORD           PUBLIC,
                    BUFFER$STATUS    BYTE           PUBLIC;

            /*******************************************************************
             *                    INSERT CHARACTER INTO FIFO                   *
             *******************************************************************/

  4   1     BUFF$IN: PROCEDURE (CHAR) PUBLIC;
  5   2     DECLARE
                    CHAR    BYTE,

  6   2     IF ((BUFFER$STATUS=EMPTY) AND ((CHAR=LF) OR (CHAR=CR)))
                THEN
  7   2           DO;
  8   3             OUTPUT(SET$INT)=HANDSHAKE$INT;  /* Enable two-wire handshake interrupt */
  9   3             BUFFER$STATUS=INUSE,
 10   3             OUTPUT(PORT$2)=0;               /* Output NULL character to get
                                                          the interrupt started */
 11   3           END,

 12   2     FIFO(IN$POINTER)=CHAR,    /* Put CHAR into FIFO and increment pointer */
 13   2     IN$POINTER=((INPOINTER+1) AND 1FFFH);

 14   2     IF (((IN$POINTER+4) AND 1FFFH)=OUT$POINTER) /* If the buffer is full, stop reception *
                THEN
 15   2           DO;  /* Send CTS FALSE, and light up buffer-full LED */
```

```
FL/M-86 COMPILER    BUFFMOD

 16   3              OUTPUT(PORT$1)=((INPUT(PORT$1) OR 40H) AND 7FH);
 17   3              BUFFER$STATUS=FULL;
 18   3             END;
 19   2      END BUFF$IN;

             /*********************************************************************
              *                    REMOVE CHARACTER FROM FIFO                     *
              *********************************************************************/

 20   1      BUFF$OUT PROCEDURE BYTE PUBLIC;
 21   2      DECLARE  CHAR   BYTE;
 22   2      CHAR=FIFO(OUT$POINTER);
 23   2      OUT$POINTER=((OUT$POINTER+1) AND 1FFFH);
 24   2      IF OUT$POINTER=IN$POINTER  /* If the buffer is EMPTY disable the output to LP */
                 THEN
 25   2            DO;
 26   3              DISABLE;
 27   3              BUFFER$STATUS=EMPTY;
 28   3              OUTPUT(RST$INT)=HANDSHAKE$INT;
 29   3              ENABLE;
 30   3            END;

             /*  If the buffer is ready to fill up again then send CTS TRUE */

 31   2      ELSE  IF ((BUFFER$STATUS=FULL) AND (((OUT$POINTER-200) AND 1FFFH)=IN$POINTER))
                     THEN
 32   2                 DO; /* Turn off buffer-full LED and turn on CTS */
 33   3                    OUTPUT(PORT$1)=((INPUT(PORT$1) AND 0BFH) OR 80H);
 34   3                    BUFFER$STATUS=INUSE;
 35   3                 END;
             RETURN CHAR;

 37   2      END BUFF$OUT;

             /*********************************************************************
              *                  SEND ERROR MESSAGE TO SERIAL PORT                *
              *********************************************************************/

 38   1      ERROR:PROCEDURE (STATUS) PUBLIC;
 39   2              DECLARE STATUS   BYTE,
                             MESSAGE  BASED  MESSAGE$PTR(1) BYTE;

 40   2              IF (STATUS AND 02H)>0
                        THEN
 41   2                   STATUS=2;
 42   2      ELSE    IF (STATUS AND 04H)>0
                        THEN
 43   2                   STATUS=3;
 44   2      ELSE    IF (STATUS AND 08H)>0
                        THEN
 45   2                   STATUS=4;
 46   2      ELSE    IF (STATUS AND 01H)>0
                        THEN
 47   2                   STATUS=1;
                     DO CASE STATUS;
 49   3
 50   3                 MESSAGE$PTR=@FRAME(0);
```

```
PL/M-86 COMPILER    BUFFMOD

 51   3                 MESSAGE$PTR=@OVER$RUN(0);
 52   3                 MESSAGE$PTR=@PARITY(0);
 53   3                 MESSAGE$PTR=@BREAK(0);
 54   3             END;

 55   2             J=1;          /* Point to second character in string */
 56   2             OUTPUT(SET$INT)=TRANSMIT$INT;
 57   2             OUTPUT(TX$BUFF)=MESSAGE(0);
 58   2      END ERROR;

             /*********************************************************************
              *                RELOAD SERIAL PORT CONFIGURE BYTE                  *
              *********************************************************************/

 59   1      PROGRAM: PROCEDURE PUBLIC;
 60   2             DECLARE TEMP    BYTE,
                            CHAR    BYTE;

 61   2             DO WHILE (INPUT(STATUS$REG) AND 40H)=0; /* Wait for next byte */
 62   3             END;

 63   2             CHAR=INPUT(RX$BUFF);

 64   2             IF CHAR=0       /* If second byte is 0, exit program mode */
                        THEN
 65   2                    DO;
 66   3                        OUTPUT(RECEIVE$TIMER)=70;
 67   3                        CALL BUFF$IN (CHAR);
 68   3                        RETURN;
 69   3                    END;

 70   2             TEMP=(CHAR AND 0FH);

 71   2             DO WHILE (INPUT(STATUS$REG) AND 40H)=0;
 72   3             END;

 73   2             TEMP=(INPUT(RXBUFF) AND 0FH) OR SHL(TEMP,4);

 74   2             SERIAL$FORMAT (SHR(PORT$PTR,3))=TEMP;
 75   2      END PROGRAM;


 76   1      END BUFF$MOD;


MODULE INFORMATION:

     CODE AREA SIZE     = 01E4H    484D
     CONSTANT AREA SIZE = 0000H      0D
     VARIABLE AREA SIZE = 200BH   8203D
     MAXIMUM STACK SIZE = 000AH     10D
     189 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS

END OF PL/M-86 COMPILATION
```

```
PL/M-86 COMPILER    PON_MOD

SERIES-III PL/M-86 V1.0 COMPILATION OF MODULE PON_MOD
OBJECT MODULE PLACED IN :F1:PON.OBJ
COMPILER INVOKED BY:  PLM86.86. :F1 PON.SRC


            $DEBUG

            /*************************************************************************
             *                                                                       *
             *      POWER ON INITIALIZATION OF THE LINE PRINTER MULTIPLEXER          *
             *                                                                       *
             *************************************************************************/

 1          PON_MOD: DO;

            $NOLIST

 3   1      DECLARE BUFFER$STATUS    BYTE     EXTERNAL,
                    IN$POINTER       WORD     EXTERNAL,
                    OUT$POINTER      WORD     EXTERNAL,
                    PORT$PTR         BYTE     EXTERNAL,
                    SERIAL$FORMAT(8)BYTE      EXTERNAL;

 4   1      POWER$ON: PROCEDURE PUBLIC;

 5   2      DECLARE I                BYTE;

 6   2      DISABLE;

                              /*  INITIALIZE THE MUART */

 7   2      OUTPUT(CMD$1)=01000011B;        /*8086 MODE, FREQ=1KHz, 1 STOP BIT, &
                                               7 BITS/CHARACTER*/
 8   2      OUTPUT(CMD$2)=10110100B;        /*ODD PARITY, SYSTEM CLOCK=1.024 MHz, &
                                               9600 bps*/
 9   2      OUTPUT(CMD$3)=01111111B;        /*CLEAR CMD$3 REGISTER*/
10   2      OUTPUT(CMD$3)=10110001B;        /*RESET, INTERRUPT ACKNOWLEDGE ENABLE, &
                                               NESTED INTERRUPT MODE*/
11   2      OUTPUT(MODE)=10000101B;         /*CASCADE TIMERS 35 FOR THE
                                               RECEIVE$TIME$OUT TIMER; BYTE OUTPUT MODE*/

12   2      OUTPUT(PORT$1$CTRL)=11111000B;  /*PORT 1: RTS=INPUT, THE REST ARE OUTPUTS*/

13   2      OUTPUT(PORT$1)=11000000B;       /*POINT TO THE FIRST PORT, CTS IS F
                                               AND BUFFER IS NOT FULL*/

                 /* INITIALIZE FLAGS, VARIABLES, AND ARRAYS */

14   2      BUFFER$STATUS=EMPTY;
15   2      IN$POINTER=0;  OUT$POINTER=0;
17   2      PORT$PTR=0;

18   2          DO I=0 TO 7;
```

```
PL/M-86 COMPILER    PON_MOD

 19   3            SERIAL$FORMAT(I)=10010100B;      /* ON POWER-UP ALL EIGHT SERIAL PORTS
                                                      DEFAULT TO 9600 bps, ODD PARITY, AND
                                                      7 BITS/CHARACTER*/
 20   3            END;
 21   2      END POWER$ON;
 22   1      END PON_MOD;


MODULE INFORMATION:

     CODE AREA SIZE      = 0058H     88D
     CONSTANT AREA SIZE  = 0000H      0D
     VARIABLE AREA SIZE  = 0001H      1D
     MAXIMUM STACK SIZE  = 0002H      2D
     98 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS

END OF PL/M-86 COMPILATION
```

APPENDIX B
LISTING OF THE WRITE PROGRAM

```
PL/M-80 COMPILER

ISIS-II PL/M-80 V4.0 COMPILATION OF MODULE WRITEMOD
OBJECT MODULE PLACED IN :F1:WRITE.OBJ
COMPILER INVOKED BY:   :F2:PLM80 :F1:WRITE.SRC


                $DEBUG
  1             WRITE$MOD:DO,

                /*********************************************************************
                 *                                                                   *
                 *    WRITE PROGRAM: READS A FILE FROM A DISK AND COPIES             *
                 *                   IT TO SERIAL CHANNEL 2 ON THE MDS               *
                 *                                                                   *
                 *    SYNTAX OF WRITE: WRITE :DEVICE:NAME.EXTENSION                  *
                 *                                                                   *
                 *********************************************************************/

  2   1         DECLARE  LIT            LITERALLY       'LITERALLY',
                         USART$DATA     LIT             '0F6H',
                         USART$STATUS   LIT             '0F7H',
                         RTS            LIT             '20H',
                         TXEN           LIT             '01H',
                         RXE            LIT             '04H',
                         CR             LIT             '0DH',
                         LF             LIT             '0AH',
                         TAB            LIT             '09H',
                         SP             LIT             '20H',
                         ESC            LIT             '1BH',
                         FORM$FEED      LIT             '0CH';

  3   1         DECLARE  AFT$IN         ADDRESS,
                         FILENAME(15)   BYTE,
                         STATUS         ADDRESS,
                         BUFFER(32000)  BYTE,
                         ACTUAL         ADDRESS,
                         CHAR$COUNT     ADDRESS,
                         BYE(42)        BYTE    INITIAL
                ('WROTE ',0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,' TO THE LINE PRINTER',0AH,0DH),
                         I              ADDRESS,
                         J              BYTE;

                /*********************************************************************
                 *                  EXTERNAL SYSTEM.LIB PROCEDURES                   *
                 *********************************************************************/

  4   1         OPEN:
                         PROCEDURE (AFTNPTR,FILE,ACCESS,MODE,STATUS) EXTERNAL;
  5   2                  DECLARE (AFTNPTR,FILE,ACCESS,MODE,STATUS) ADDRESS;
  6   2         END OPEN;

  7   1         READ:
                         PROCEDURE (AFTN,BUFFER,COUNT,ACTUAL,STATUS) EXTERNAL;
  8   2                  DECLARE (AFTN,BUFFER,COUNT,ACTUAL,STATUS) ADDRESS;
  9   2         END READ;

 10   1         WRITE:
```

PL/M-80 COMPILER

```
                    PROCEDURE (AFTN,BUFFER,COUNT,STATUS) EXTERNAL;
 11   2             DECLARE (AFTN,BUFFER,COUNT,STATUS) ADDRESS;
 12   2       END WRITE;

 13   1       CLOSE:
                    PROCEDURE (AFTN,STATUS) EXTERNAL;
 14   2             DECLARE (AFTN,STATUS) ADDRESS;
 15   2       END CLOSE;

 16   1       ERROR:
                    PROCEDURE (ERRNUM) EXTERNAL;
 17   2             DECLARE (ERRNUM) ADDRESS;
 18   2       END ERROR;

 19   1       EXIT:
                    PROCEDURE EXTERNAL;
 20   2       END EXIT;

               /*********************************************************************
                *              WAIT UNTIL USART TRANSMITTER IS READY                *
                *********************************************************************/

 21   1       TXRDY:
                    PROCEDURE;
 22   2             DO WHILE ( (INPUT(USART$STATUS) AND 01H) = 0 );
 23   3             END;
 24   2       END TXRDY;

               /*********************************************************************
                *                        BEGIN MAIN PROGRAM                         *
                *********************************************************************/
 25   1       BEGIN:
                    STATUS=0;

 26   1             CALL READ(1,.FILENAME,15,.ACTUAL,.STATUS);  /* Read in file and path name */

 27   1             IF STATUS <> 0
                         THEN
 28   1                  GO TO DONE;

 29   1             CALL OPEN(.AFT$IN,.FILENAME,1,0,.STATUS);  /* Open up the file */

 30   1             IF STATUS <> 0
                         THEN
 31   1                  GO TO DONE;

 32   1       REPEAT:
                    CALL READ(AFT$IN,.BUFFER,32000,.ACTUAL,.STATUS);

 33   1             IF STATUS <> 0
                         THEN
 34   1                  GO TO DONE;

 35   1             CHAR$COUNT=0;  /* CHAR$COUNT keeps track of the tab columns in each line */

 36   1             OUTPUT(USART$STATUS)= RTS OR TXEN;
```

PL/M-80 COMPILER

```
37   1          IF BUFFER(0)=FORM$FEED    /* If the first character is a form feed
                                              remove it. Form feeds are inserted at the
                                              end of a file */
                    THEN
38   1                 DO,
39   2                    BUFFER(0)=00H;
40   2                    CHAR$COUNT=-1,
41   2                 END,

42   1          DO I = 0 TO (ACTUAL - 1);

43   2                 IF  (BUFFER(I)=TAB)  /* Replace TAB characters with the
                                                 appropriate number of spaces */
                          THEN
44   2                       DO,
45   3                            CALL TXRDY;
46   3                            OUTPUT(USART$DATA)=SP;
47   3                            CHAR$COUNT=CHAR$COUNT+1;

48   3                            DO WHILE ((CHAR$COUNT AND 0007H)<>0);
49   4                                 CALL TXRDY;
50   4                                 OUTPUT(USART$DATA)=SP;
51   4                                 CHAR$COUNT=CHAR$COUNT+1,
52   4                            END;
53   3                       END;
                    ELSE
54   2                     IF BUFFER(I)=ESC /* If outputting ESC, then output a
                                                0 next so the LPM does not get
                                                    re-programmed */
                              THEN
55   2                           DO J=0 TO 1;
56   3                              CALL TXRDY;
57   3                              OUTPUT(USART$DATA)=0;
58   3                           END;
                    ELSE /* If the character is not an ESC or TAB then output it */
59   2                 DO;
60   3                   CALL TXRDY;
61   3                   OUTPUT(USART$DATA)=BUFFER(I);
62   3                   IF (BUFFER(I)>1FH AND BUFFER(I)<>7FH)

                            THEN                    /* Only increment CHAR$COUNT
                                                       for printable characters */
63   3                         CHAR$COUNT=CHAR$COUNT+1;

64   3                   IF ( (BUFFER(I)=CR) OR (BUFFER(I)=LF) )
                            THEN                /* Reset CHAR$COUNT for CR or LF */
65   3                         CHAR$COUNT=0,
66   3                 END;
67   2          END,

68   1          IF ACTUAL = 32000 /*If the file is more than 32K, get some more data */
                    THEN
69   1                 GO TO REPEAT;

70   1          CALL TXRDY;              /* Terminate file with CR, LF, and FF */
71   1          OUTPUT(USART$DATA)=CR;
72   1          CALL TXRDY;
```

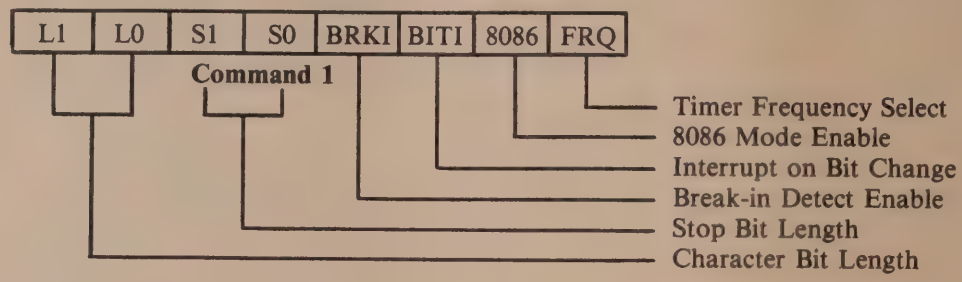
PL/M-80 COMPILER

```
 73   1                OUTPUT(USART$DATA)=LF;
 74   1                CALL TXRDY;
 75   1                OUTPUT(USART$DATA)=FORM$FEED;

 76   1                OUTPUT(USART$STATUS)=RXE OR TXEN; /* Shut off RTS */

 77   1                CALL CLOSE (AFT$IN,.STATUS);

 78   1                DO I=0 TO 14;              /* Output sign off message */
 79   2                     IF FILENAME(I)=CR
                              THEN
 80   2                         GO TO SKIP;
 81   2                     BYE(I+5)=FILENAME(I);
 82   2                END;

 83   1        SKIP:   CALL WRITE(0,.BYE,42,.STATUS);

 84   1                GO TO NEXT;

 85   1        DONE:
                        CALL ERROR(STATUS);

 86   1        NEXT:
                        CALL EXIT;

 87   1        END WRITE$MOD;


MODULE INFORMATION:

     CODE AREA SIZE     = 0209H    521D
     VARIABLE AREA SIZE = 7D44H  32068D
     MAXIMUM STACK SIZE = 0008H      8D
     191 LINES READ
     0 PROGRAM ERRORS

END OF PL/M-80 COMPILATION
```

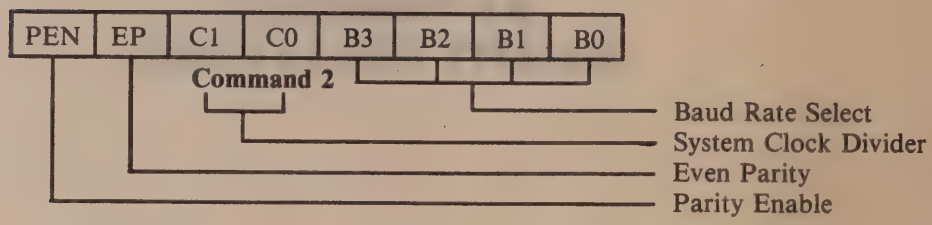
APPENDIX C
MUART REGISTERS

8085 Mode: AD3 AD2 AD1 AD0
8086 Mode: AD4 AD3 AD2 AD1

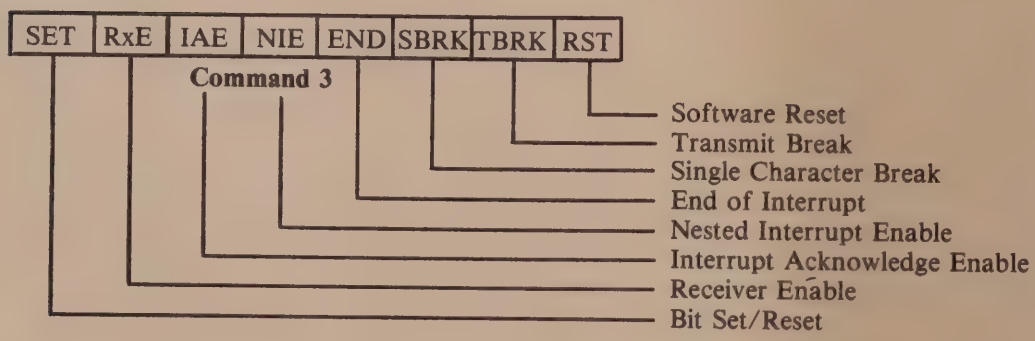
0000



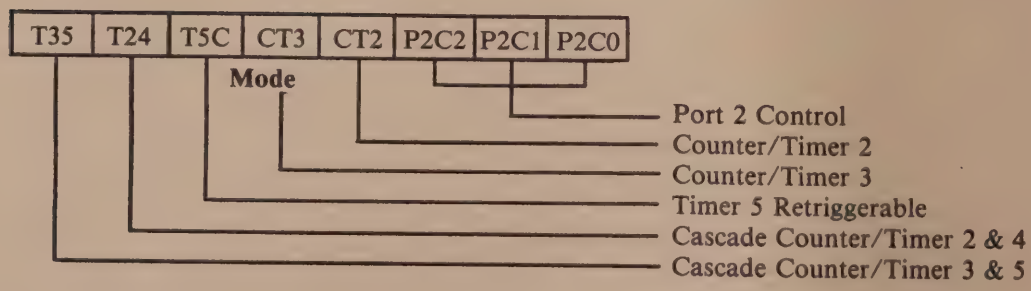
0001



0010



0011



0100

P17	P16	P15	P14	P13	P12	P11	P10

Port 1 Control

Output/$\overline{\text{Input}}$ of Port 1 pins

(Write only)

0101

L7	L6	L5	L4	L3	L2	L1	L0

Set Interrupts

Enable

(Write only)

0110

L7	L6	L5	L4	L3	L2	L1	L0

Reset Interrupts

Disable

(Read only)

0101

L7	L6	L5	L4	L3	L2	L1	L0

Interrupt Enable

Interrupt Levels Enabled

(Read only)

0110

0	0	0	D4	D3	D2	0	0

Interrupt Address

Interrupt Level in Service

(Write only)

1111

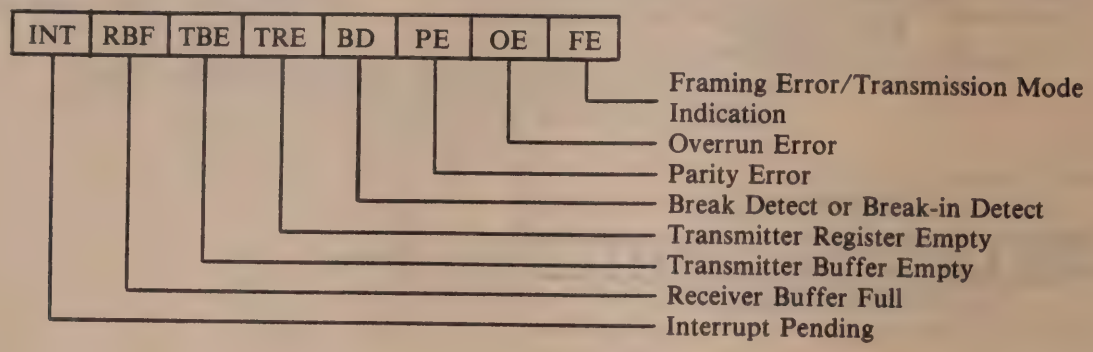
0	RS4	RS3	RS2	RS1	RS0	TME	DSC

Modification

Disable Start Bit Check
Transmit Mode Enable
Receiver Sampling Point

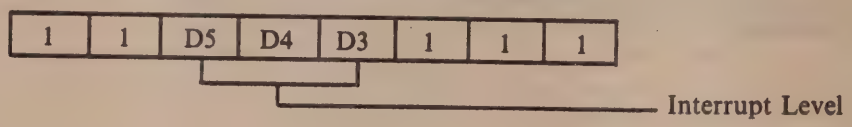
Status Register (Read only)

1111

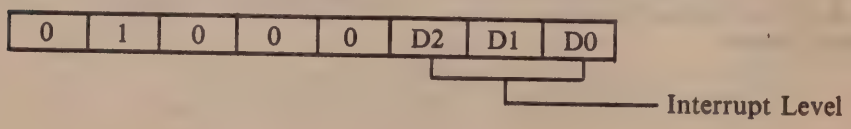


Response to INTA

8085-Mode (RST-instruction in response to $\overline{\text{INTA}}$)



8086-Mode (Interrupt Vector in response to second $\overline{\text{INTA}}$)



intel

APPLICATION NOTE

AP-183

August 1984

8256AH Multifunction Peripheral Simplifies Microcomputer I/O Design

CHRISTOPHER SCOTT

© Intel Corporation, 1984

Order Number: 231125-001

8256AH Multifunction Peripheral Simplifies Microcomputer I/O Design

CONTENTS

INTRODUCTION

Description of the 8256AH

HARDWARE DESCRIPTION

8256AH/80186 System Design

RS-232C Hardware Interface

Parallel I/O with Handshaking

SOFTWARE DESCRIPTION

Serial RS-232C Interface

RS-232C Control Signals Interrupt Structure

CONCLUSION

APPENDIX A.

Software Listing

FIGURES

1a. System Block Diagram Without 8256AH

1b. System Block Diagram With 8256AH

2. 8256AH Internal Block Diagram

3. 8256AH / 80186 Schematic

4. Block Diagram of the 8256AH Serial RS-232C Interface Software Structure

5. 8256AH Interrupt Source To Priority Level Map

6. Port 1 RS-232C Pin Definition

7. Receive Data Interrupt Service Routine Software Flowchart

8. Transmit Data Interrupt Service Routine Software Flowchart

Additional Sources of Information

Ap Note 153 Designing with the 8256AH

INTRODUCTION

A primary goal of microcomputer system design is to provide the required functionality and flexibility with the fewest number of components. The 8256AH Multifunction Peripheral is designed specifically to meet these conflicting requirements. Four of the most common microcomputer system functions, previously requiring up to four separate MSI or LSI devices, are combined into one LSI device. The 8256AH incorporates a serial asynchronous communication channel, two 8-bit parallel I/O ports, five 8-bit timer/counters and an eight level priority interrupt controller in one 40 pin package. Its flexible design allows it to directly interface to most microprocessors, including Intel's MCS-85, iAPX-86, iAPX-88, iAPX-186 and iAPX-188, and the MCS-48 and MCS-51 family of single-chip microcomputers.

This application note describes using the 8256AH to implement a Data Terminal Equipment (DTE) RS-232C serial asyncrhonous communication link with the control signals necessary to interface to a Bell 103/212A modem. The interface requires a total of nine interface signals. Three of these signals, TxD, RxD and CTS, are provided by the UART section of the 8256AH. The balance of the RS-232C interface signals are implemented using six of the independently programmable parallel PORT 1 lines. In addition, the application design provides an eight bit parallel I/O port with handshaking signals. The on-chip priority interrupt controller enables the RS-232C serial interface an the parallel interface to operate on an interrupt basi The 8256AH uniquely addresses the complexities of implementing an RS-232C communications interface. By utilizing the built-in hardware and software features of the 8256AH, the design achieves flexibility with simplicity, qualities often exclusive of one another.

Previous solutions required four components to implement the same interface. Figure 1 illustrates the basic system block diagrams for the two solutions. In Figure 1a the 8251A Programmable Communications Interface provides the UART serial communications interface. The 8254 Programmable Interval Timer provides baud rate generation and other timing functions, such as time-out loops, needed for software support of an RS-232C interface. These are especially needed if the RS-232C channel is to operate in an interrupt system environment. The 8255A Programmable Peripheral Interface provides parallel I/O with one port dedicated to the RS-232C control signals. The 8259A Priority Interrupt Controller provides an eight level priority interrupt structure. This represents a total of 120 device pins compared to the single 40 pin 8256AH, and 465 mA current requirement verses a 160 mA current requirement. Figure 1b represents the 8256AH solution incorporating the four functions in one package.

In some data communication applications only three lines - ground, Transmit Data and Receive Data - are used for serial communication. An example is communication between an ASCII terminal or printer and a personal computer. These devices are usually located close to one another and in general do not require the additional control signals of the EIA RS-232C serial communications standard. In other data communications applications, this same equipment requires that the integrity of the serial communications link be constantly monitored. This enables the host system to control the data trransmission at all times, whether it be a host computer or intelligence local to a communications device, such as an ASCII terminal. The need for control and monitoring of the serial line is particularly important when the communications link is over telephone lines using a modem. In a Switched Network, where a number of serial devices share the same communications line, the control signals are crucial to the system's multiplexing the single line.

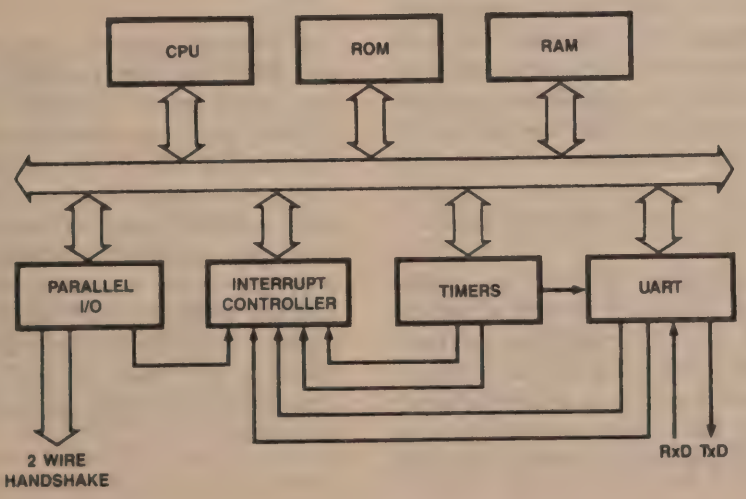


Figure 1a. System Block Diagram Without the 8256AH

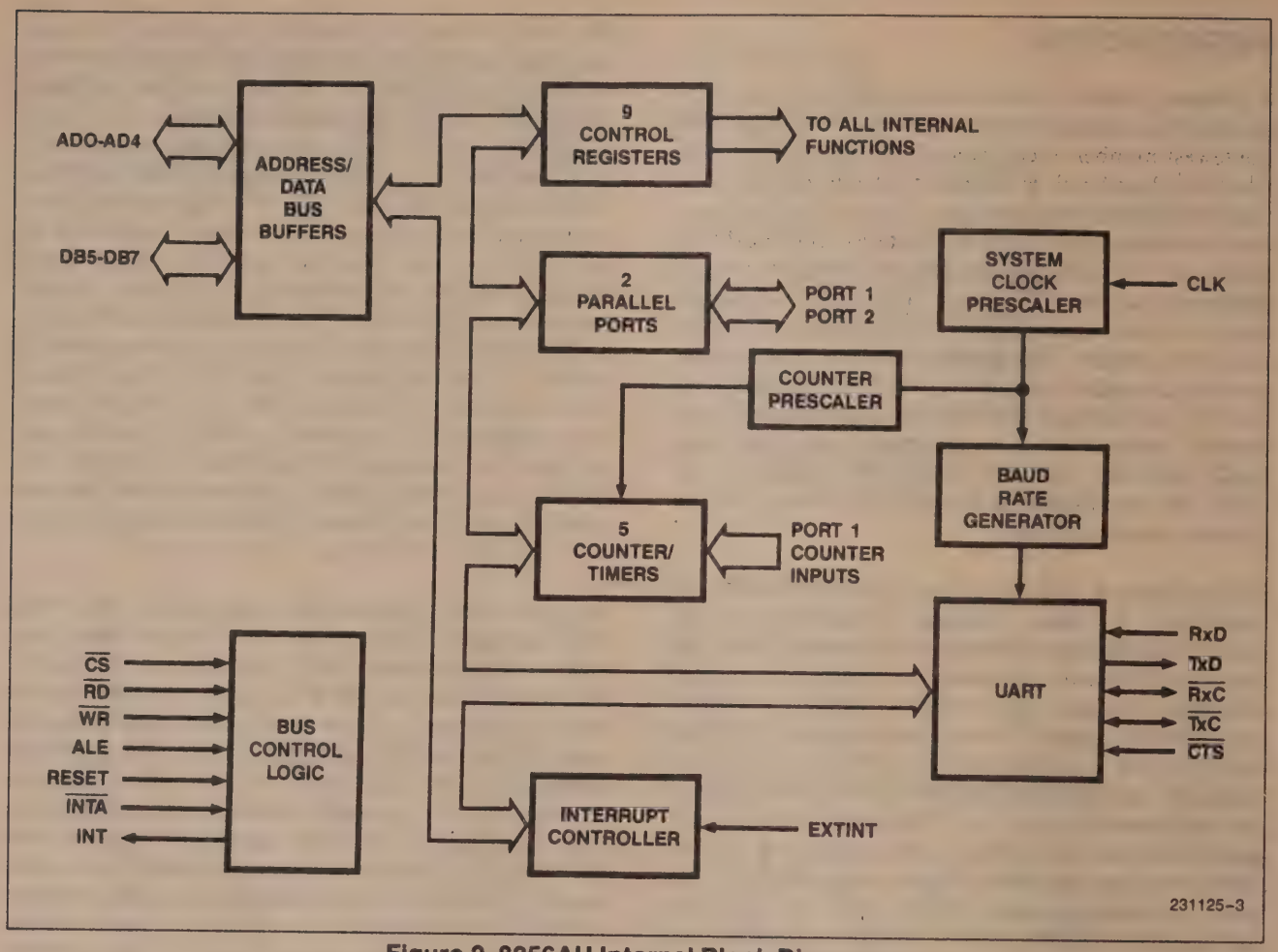


Figure 2. 8256AH Internal Block Diagram

parallel port with ACK/OBF and STB/IBF two wire handshake signals. In the latter configuration, the six remaining I/O lines may be used as either independently programmable I/O lines, or as predefined special function inputs and/or outputs, such as a second external interrupt input or timer/counter inputs.

The five 8-bit programmable timer/counters are binary presettable downcounters. In addition, an independent on-chip Baud Rate Generator is provided for the UART. The clock sources for the timers/counters may be either internal or external - via programmed parallel port pins - depending upon whether they are configured as timers or counters. Four of the timer/counters may be cascaded to form two 16-bit timer/counters. Each of the five timer/counters has its own read/write register.

The eight level priority interrupt controller has twelve possible interrupt sources. Ten of the sources are internal and two are external. One of the external interrupt sources is a fixed pin; EXTINT. The second is one of the parallel Port 1 pins which can be programmed as an external interrupt source. The twelve interrupt sources are internally mapped to the eight interrupt priority levels.

The interrupt controller may be programmed to operate in either a Normal or Nested Interrupt Mode. In Normal Mode any interrupt may interrupt any other interrupt based upon the enable/disable bits in the Interrupt Enable, or Mask, Register. In the Nested Mode only an interrupt of higher piority may interrupt one of lower priority, again based upon the bits in the Enable Register.

The 8256AH interrupt structure supports both 8085 and 8086 interrupt vectoring methods via the INTR and INTA signals. In vectored interrupt operation the 8256AH places the interrupt vector address on the data bus during the INTA sequence. In addition the 8256AH supports non-vectored interrupt interfaces, such as MCS-51 and MCS-48 systems. In non-vectored interrupt applications the host system simply reads the interrupt vector address from the Interrupt Address Register of the 8256AH. Reading the interrupt address register clears the INT pin and acknowledges that the interrupt has been serviced. This is the functional equivalent to an INTA sequence generated by the host processor.

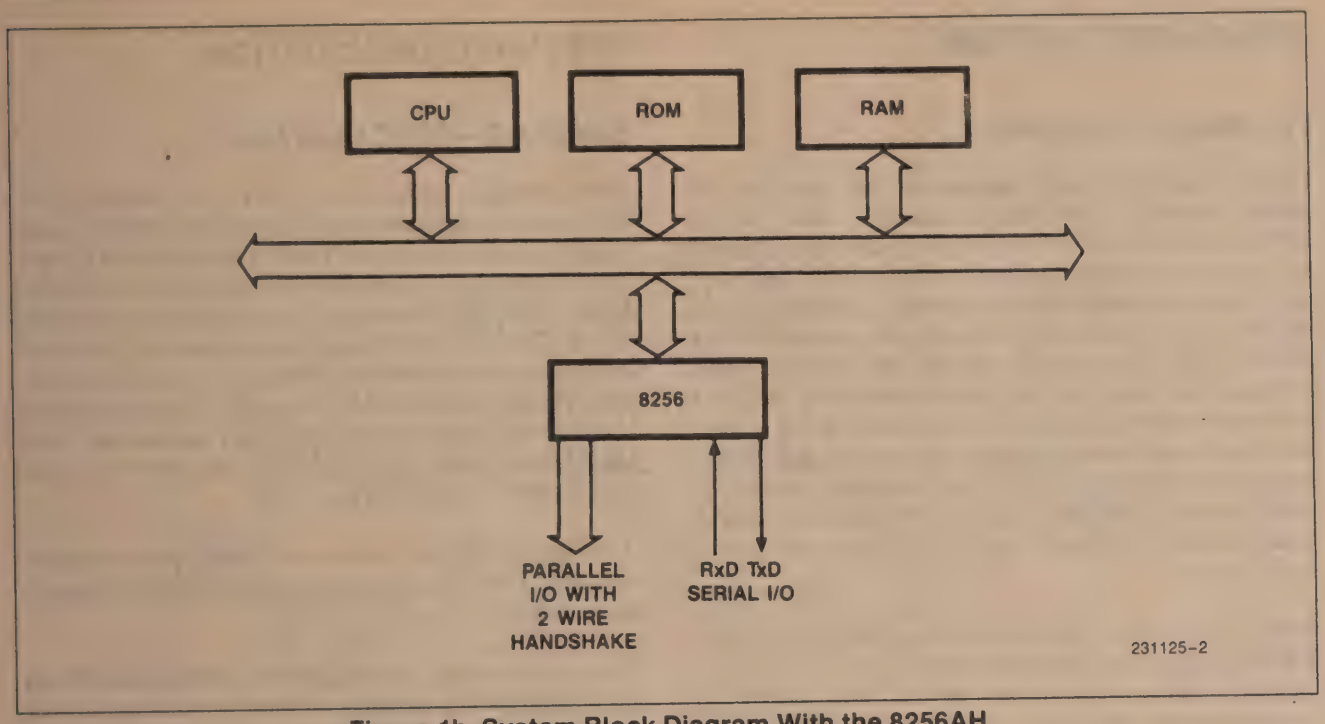


Figure 1b. System Block Diagram With the 8256AH

This Application Note assumes that the reader is familiar with the 8256AH Data Sheet and with the RS-232C communication protocol and terminology. A complete software listing is provided in Appendix A. A complete description and definition of the RS-232C interface standard may be found in the book "Data Communications: A Users Guide" by Kenneth Sherman, Reston Publishing 1981.

DESCRIPTION OF THE 8256AH

The 8256AH combines four commonly used peripheral functions into one device (see Figure 2);

1. A full-duplex, double-buffered serial asynchronous Receiver/Transmitter (UART) with an on-chip Baud Rate Generator.

2. Two 8-bit parallel I/O ports; One bit programmable, One nibble programmable.

3. Five 8-bit timer/counters; 4 can be cascaded to form 2 16-bit timer/counters

4. An 8-level priority interrupt controller.

The 8256AH uses the standard bus control signals compatible with Intel's family of peripherals and microprocessors. The microprocessor interface utilizes a multiplexed address/data bus. Four of the eight address/data lines are used to generate the register address. This enables all of the 8256AH's functionality to be contained in a 40 pin package while retaining direct register addressing.

The sixteen directly addressable internal read/write registers provide control for all of the 8256AH's various functions. Fourteen of the registers are read/write, one, the Status Register, is read only and one, the Modification Register, is write only. Three Command Registers configure the operating environment including the type of CPU, 8 or 16 bit, and system clock frequency. Command Register Three provides bit set-reset capability for control of such functions as End of Interrupt, Nested Interrupts, Interrupt Acknowledge and UART Receive Enable. The Status Register provides all information about the UART's transmitter and receiver, and the state of the interrupt (INT) output pin to the microprocessor. The Mode Register defines the configuration of the two parallel ports and the five timer/counters. The write only Modification Register is used to alter two standard functions of the receiver, start bit sampling and to enable a special indicator flag for half-duplex operation. In addition, six registers control the two parallel ports. Two registers provide for UART Transmit and Receive Buffers. Ten registers are used for timer/counter interface, and four registers provide for Priority Interrupt Controller support.

The UART section of the 8256AH features a full-duplex double-buffered transmitter and receiver with separate control registers. The internal baud rate generator provides the thirteen common sampling rates from 50 bps to 19.2 kbps. An external baud rate clock can also be used, with programmable choice of 1X, 32X or 64X sampling rates.

The two parallel I/O ports can be configured as two independent 8-bit parallel I/O ports, or as one 8-bit

DESIGN DESCRIPTION

Hardware Description

Figure 3 shows a block diagram of this application's system design. The microprocessor used is an iAPX-186 with two 8256AH's for parallel and serial I/O, as well as for providing a variety of system support functions. One 8256AH is used to implement both the RS-232C modem interface and provide multiplexed parallel I/O. The system uses the Intel 957B System Monitor for control of the system hardware and software development support. The second 8256AH is used for basic serial communication between an ASCII terminal and the Intel 957B System Monitor residing in 16K bytes of EPROM. The two 8256AHs provide a total of six I/O channels - two UARTs and four parallel I/O ports.

When one of the 8256AHs is configured for the serial RS-232C interface, one of its parallel ports, Port 1 pins 2-7, provides control signals for the serial interface. Four of the RS-232C control signals (CTS, DSRS, DSR and CD) are OR'd to the EXTINT pin of the 8256AH. If any of these signals change from their defined state, an interrupt is generated to the 8256AH. The modem driver software then responds to the interrupt by reading the Port 1 register, determines the signal generating the interrupt and responds accordingly (see the software listing; INT—MOD). In addition to the RS-232C control signals, the communications software can support all of the standard UART error conditions such as framing errors, underrun, overrun and parity, if parity is enabled.

Parallel I/O With Handshaking

The remaining two Port 1 lines, not used for the RS-232C control signals, provide ACK/OBF and STB/IBF handshaking signals for parallel Port 2. In an environment which utilized the second parallel port, while implementing the above described RS-232C channel, both would operate on an interrupt basis. The interrupt software algorithm depends upon whether the parallel port is configured as input or output, and whether Nested or Normal interrupt mode is programmed. If Nested Interrupt Mode is used, the software flow would default to parallel input or output (as programmed) with Port 2 handshaking the lowest priority interrupt. The serial channel would then interrupt parallel Port 2 transmission whenever the serial channel transmitted or received a character. The RS-232C control signals, OR'd to the External Interrupt (EXTINT) pin, would have the highest interrupt controller priority. The Software Description below describes this in greater detail.

SOFTWARE DESCRIPTION

Serial RS—232C Interface

The software is written in PL/M and is broken up into four separate modules, each containing several procedures. A block diagram of the software structure is given in Figure 4. The modules are identified by the dotted boxes, and the procedures are identified by the solid boxes. Two or more procedures connected by a solid line means the procedure above calls the procedure below. The procedures without any solid lines connecting them are interrupt procedures. They are entered when the 8256AH interrupts the 80186 and vectors an indirect address to the 80186.

The Serial RS-232C Interface software uses nested interrupts. The priority of the interrupt procedures is given in Figure 5.

The priority of the interrupts is not programmable but they are logically oriented so that for this application the priority is correct. The serial receiver should have the highest priority since it could have overrun errors. Therefore the RxD request can interrupt any other interrupt service routine thus preventing any possibility of an overrun error.

The Serial RS-232C Interface software is entered via a GO instruction from the 957B System Monitor console. The software first calls POWR—ON—INIT which initializes the 8256AH. This sets the 8256AH to 8086 Mode with parallel Port 2 in two wire handshake mode using Port 1 pin 0-1 for Port 2 handshaking. The initialization configures six of the Port 1 lines, pins 2-7, for RS-232C handshaking—input or output depending upon the specific signal tied to the pin. Figure 6 illustrates the definition of each Port 1 RS-232C handshaking line and its direction.

Both the Serial RS-232C Interface and the parallel interface with handshaking operate on an interrupt basis. Following initialization the software enters an endless loop and awaits an interrupt from one of three sources; Receive Data (RxD), Transmit Data (TxD) or the parallel interface. In the serial interface idle state, neither transmitting nor receiving data, the software is constantly responding to TxD interrupts; a result of the Transmit Buffer (TBE) and/or Transmit Register (TRE) being continuely empty. When data is received by the RS-232C channel the RxD interrupt, being of higher priority, asserts its interrupt.

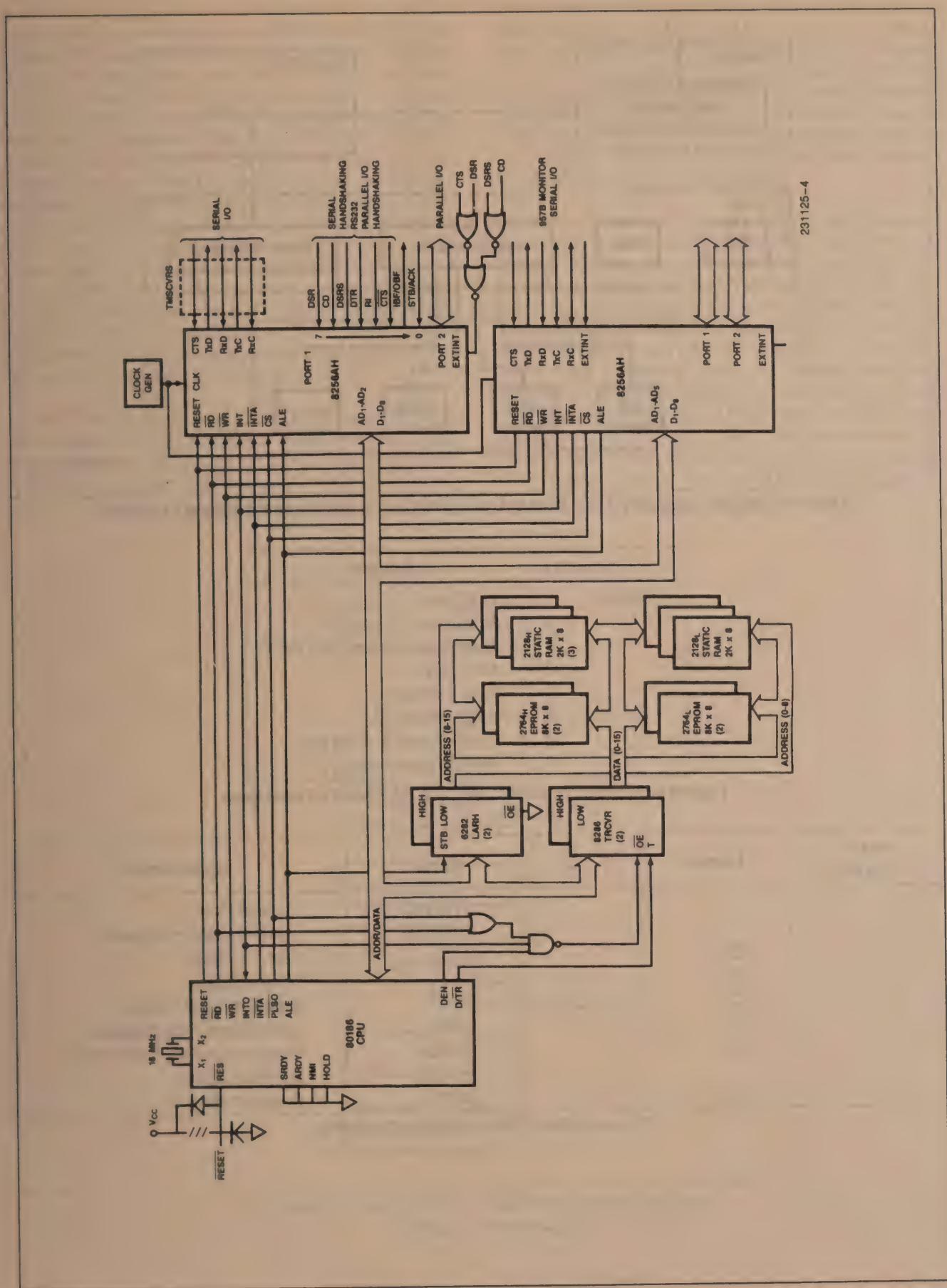


Figure 3. 8256AH / 80186 Schematic

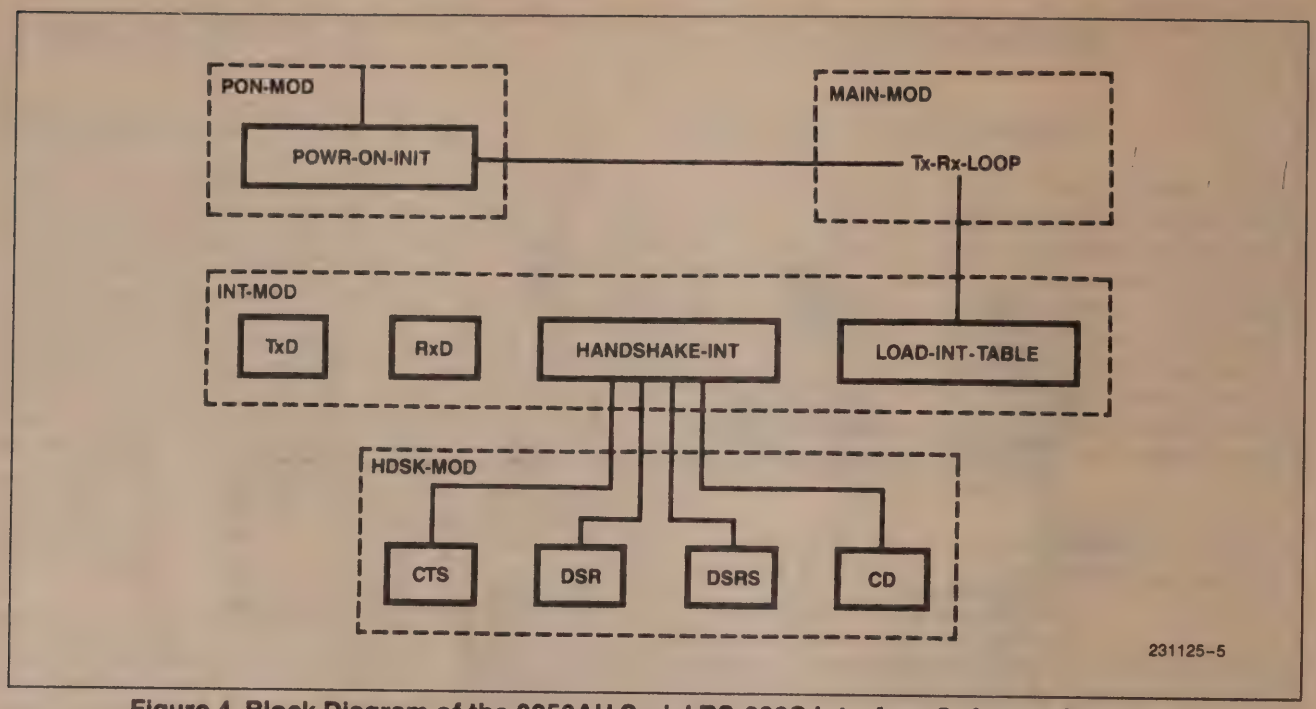


Figure 4. Block Diagram of the 8256AH Serial RS-232C Interface Software Structure

Priority		Source
Highest	0	Not Used
	1	Not Used
	2	External Interrupt (EXTINT)
	3	Not Used
	4	RxD Interrupt
	5	TxD Interrupt
	6	Timer 2 or 2 & 4 (16 bit)
	7	Port 2 Handshaking

Figure 5. 8256AH Interrupt Source To Priority Level Map

Port 1 Pin No.	Circuit	I/O	Abrev.	Signal Name
0			STB/ACK	Parallel Port 2
1			IBF/OBF	Handshaking Signals
2	CG	I	CTS	Clear To Send
3	CE	I	RI	Ring Indicator
4	CD	O	DTR	Data Terminal Ready
5	CI	I	DSRS	Data Signal Rate Selector
6	CF	I	RLSD (or CD)	Receive Line Signal Detector (Carrier Detect)
7	CC	I	DSR	Data Set Ready

Figure 6. Port 1 RS-232C Pin Definition

Although the parallel interface software is not implemented in the software listing of Appendix A, the algorithm for implementing multiplexed parallel and serial I/O is to input or output data on the parallel port during the relatively lengthy time required for serial communication overhead. The algorithm differs slightly during the serial channel idle state when the software responds to repetitive TxD interrupts. In this case the endless loop would detect the idle state repetitive TxD interrupts and disable the TxD interrupt for a short time while the parallel inputs or outputs data. This would require using one of the 8256AH timers to time out repetitive TxD interrupts. The timer used has to be lower in priority than the RxD interrupt to guarantee protection against overrun errors. Timer 2, or 2 and 4 cascaded if longer time delays are desired, provides the proper interrupt level as shown in Figure 5.

Figure 7 shows the Receive Data (RxD) interrupt service routine software flowchart. Since two conditions can generate an RxD Interrupt the Software first reads the Status Register and checks for the Break Detect (DB) bit being set. If the BD bit is clear, no Break condition being present, the data byte is read, stripped to seven bits, for an ASCII character, and sent to the system console via a call to the 957B System Monitor Console Output (CO) routine. Upon return from the 957B monitor call an End Of Interrupt (EOI) is sent to the 8256AH to reset the currently served interrupt level bit in the Interrupt Service Register.

Figure 8 shows the Transmit Data (TxD) interrupt service routine software flowchart. There are three conditions which may cause a TxD Interrupt; TBE, TRE and Break-In Detect. The TxD service routine first reads the Status Register to determine if the interrupt source is the TBE (Transmit Buffer Empty), if not then the interrupt service routine returns to the MAIN—MOD loop. If TBE = 1 (true) then a data byte is read from the 957B System Monitor Console Input (CI) routine. If the data byte is an ASCII character it is written to the 8256AH Transmit Buffer. The software exists via an EOI (End Of Interrupt) command to the 8256AH then returns to the MAIN—MOD Rx—Tx—Loop.

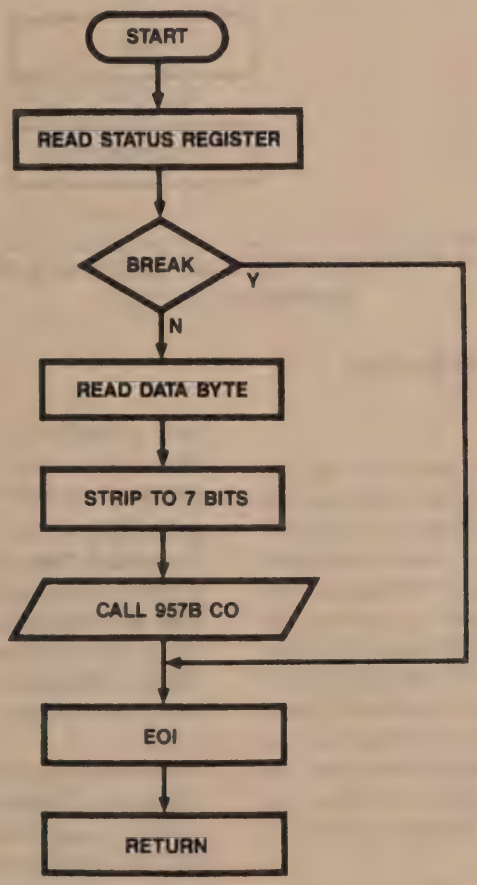


Figure 7. Receive Data Interrupt Service Routine Software Flowchart

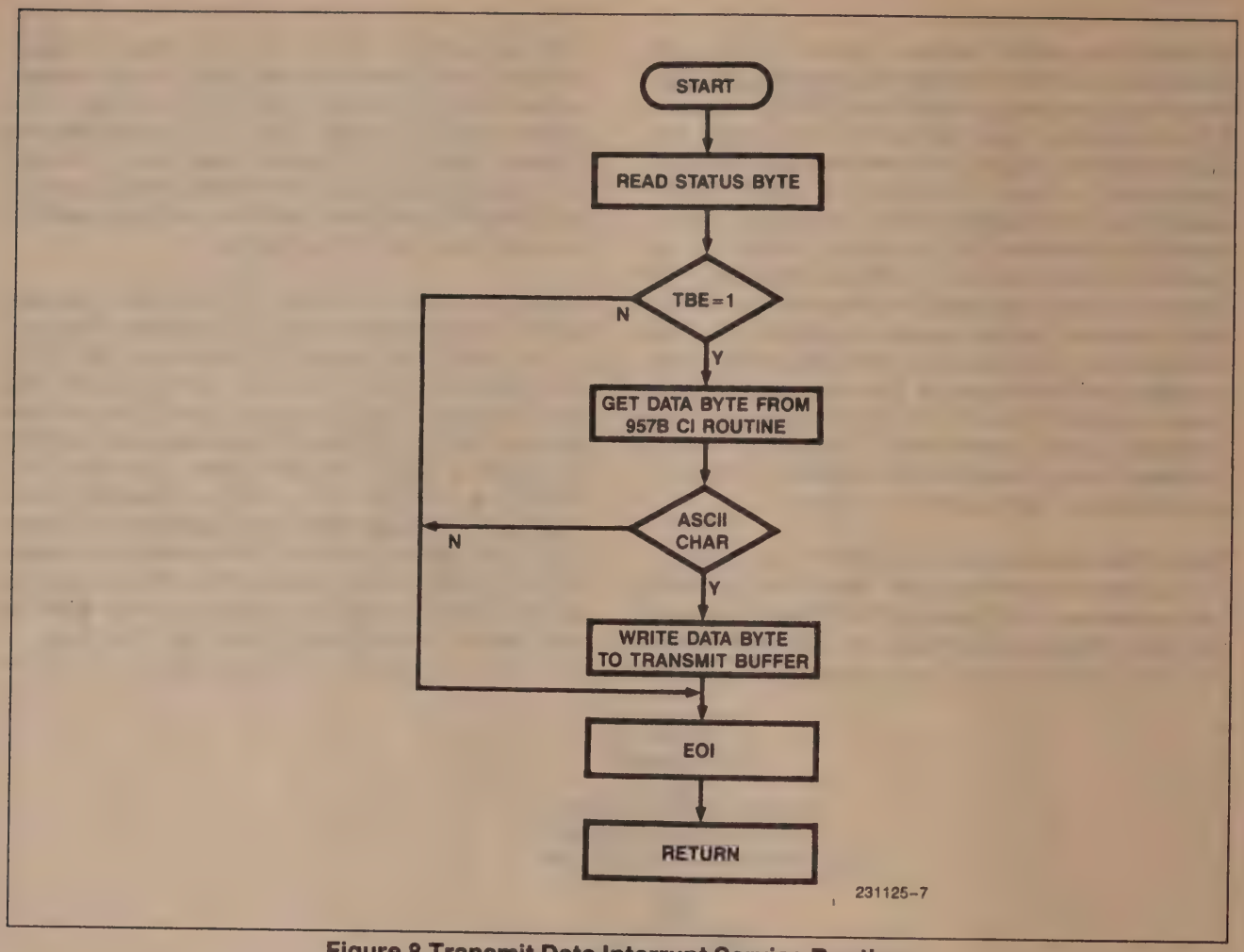


Figure 8.Transmit Data Interrupt Service Routine Software Flowchart

RS-232C Control Signals Interrupt Structure

The overall interrupt scheme is such that a change in a RS-232C handshake line causes an interrupt via the EXTINT pin on the 8256AH (see Figure 3 8256AH/80186 Schematic). The EXTINT interrupt is of higher priority than either the RxD or TxD interrupt. This enables the RS-232C handshake signals to manage the receipt or transmission of data via the nested interrupt mode of the 8256AH. The EXTINT interrupt service routine first reads the Port 1 pins 2-7 data and compares it to default state for the signal requiring service. The EXTINT interrupt service routine then calls the appropriate handshake signal service procedure as shown in the bottom module of Figure 4 Software Structure Block Diagram.

Each of the individual RS-232C control signal service procedures displays a message on the 957B monitor console device indicating the signal requiring a response. The service procedure then either initiates specific predefined actions or prompts the user with options. In a system which utilized file storage, such as a personal computer, the RS-232C software driver could pass a flag to the communications software, rather than a message. The communications software would in turn perform the same types of action but could also protect disk buffering files which might be open at the time of the interrupt. Two examples of the RS-232C Control Signal interrupt service routines, CTS and DSRS, are described below;

If Clear To Send (CTS) changes state, the UART automatically disables the transmitter. The CTS interrupt service procedure initializes the 8256AH's internal Timer. If the timer times out before CTS goes active again an interrupt is generated, a second message is displayed at the 957B monitor console prompting the user that the CTS line remains inactive. The options available at this point are to wait again, re-initializing Timer 1, or to disconnect the RS-232C channel.

If Data Signal Rate Selector (DSRS) changes state, the software prompts the user with a message that the Data Rates of the two RS-232C channels are not the same and the user is given the option of altering the data rate. This application example was interfaced to a 103A/212 Bell modem and as such prompts the user to select between 300 or 1200 bps data rates. In the case of a non-modem interface the routine could prompt the user for one of the thirteen standard data rates. The software then returns to the TxD/RxD software loop. The balance of the interrupt service procedures for the RS-232C handshaking signals function in a similar manner.

Depending upon the specific system design and software requirements, a variety of enhancements could be added to the system design. These could include interrupt traps that initiate specific corrective options or cascading multiple 8256AHs each with an RS-232C interfaces as described above. An example of an interrupt trap might be auto redial upon time out for lack of Carrier Detect (CD) upon initiating a communications link, or automatic disk file update when a receive buffer approaches overflow.

The ability of the 8256AH to be reprogrammed to meet the changing requirements of a system simplifies the overall system design and multiplies its capabilities. A simple reinitialization sequence could reconfigure the 8256AH as a UART with two parallel ports or utilize any of the various special functions of the parallel Port 1; e.g., an external timer input or an additional external interrupt input, etc. The reinitialization could also configure the 8256AH Multifunction Peripheral for a variety of custom applications.

CONCLUSION

The functional integration of the 8256AH makes it ideal for designs which require maximum flexibility and simplicity of implementation. The implementation of the RS-232C serial channel modem interface and multiplexed parallel I/O described in this application note represent a level of efficiency in peripheral performance and design previously unavailable. The 8256AH Multifunction Peripheral represents a savings of two-thirds the board space and power required by the previous four chip solution, with the added benefit of increased system reliability. The application note demonstrates the ease of implementing the variety of I/O capabilities and system support functions of the 8256AH. The integration of four common microprocessor system functions into one VLSI device enables the designer to devote valuable resources to adding features to enhance the system design, adding performance and flexibility, and reducing the system's overhead.

APPENDIX A.

SOFTWARE LISTING

```
PL/M-86 COMPILER    MAINMOD

SERIES-III PL/M-86 V2.3 COMPILATION OF MODULE MAINMOD
OBJECT MODULE PLACED IN :F2:56.OBJ
COMPILER INVOKED BY:  PLM86.86 :F2:56


            /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
            *                                                                   *
            *          8256AH MULTIFUNCTION PERIPHERAL SIMPLIFIES               *
            *             MICROCOMPUTER I/O DESIGN                              *
            *                                                                   *
            *             Intel Corporation                                     *
            *             3065 Bowers Avenue                                    *
            *             Santa Clara,  Ca.  95051                              *
            *                                                                   *
            *                                                                   *
            *                 Written By      Christopher Scott                 *
            *                                                                   *
            * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

            $MOD186 DEBUG LARGE
  1         MAINMOD:
            DO;

            /* - - -
```

```
                Intr4           lit             'pcs1 + 08h',

                Int_Reset       lit             '88h',
                SioTxEn         lit             '10h',
                SioTxRdy        lit             '20h',
                SioRxRdy        lit             '40h',
                Break           lit             '04h',
                DisIntr         lit             '00h',
                StripTo7fh      lit             '7fh',
                Port1_Strip     lit             '0fcH',

                Cmd1            lit             '43h',
                Cmd2A           lit             '07h',
                Cmd2B           lit             '09h',
                Cmd3Clr         lit             '7fh',
                Cmd3            lit             '0a1h',
                Mode            lit             '00h',
                EnRcvr          lit             '0c0h',

                A               lit             '41h',
                B               lit             '42h',
                DSR             lit             '80h',
                DSR_Flag        lit             '80h',
                CD              lit             '40h',
                CD_Flag         lit             '40h',
                DSRS            lit             '20h',
                DSRS_Flag       lit             '20h',
                DTR             lit             '10h',
                RI              lit             '08h',
                CTS             lit             '04h',
                CTS_Flag        lit             '04h',

                (Status,
                 Hndshk_Pins,
                 J)             Byte,

                Char            Byte            External,

                Message_Ptr     Pointer;

        /* - - -
```

```
                /*                                                                  */
                /*          MCO:    957B Monitor Console Output Routine             */
                /*                                                                  */
                /*          MCI:    957B Monitor Console Input Routine              */
                /*                                                                  */
                /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

  4   1         MCO: Procedure(Char) External;
  5   2           DCL        Char        Byte;
  6   2         End MCO;

  7   1         MCI: Procedure Byte External;
  8   2         End MCI;

                /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */


                /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
                /*      Initialize 8256AH Procedure                                 */
  9   1         Init56: Procedure;

 10   2           Disable;
                  /*    Output 8256AH Init Data          */
 11   2           Output(Cmd1Reg)=Cmd1;          /* 8086 mode, freg=1khz, 1 stop bit,
                                                    and 7 bit char */
 12   2           Output(Cmd1Reg)=Cmd2A;          /* odd parity, system clk=1.024mhz,
                                                    and 1200 bps */
 13   2           Output(Cmd1Reg)=Cmd3Clr;       /* clear cmd reg 3 */
 14   2           Output(Cmd1Reg)=Cmd3;          /* reset, itr ack enabled, nested
                                                     intr mode  */
 15   2           Output(Cmd1Reg)=EnRcvr;        /* enable receiver */
 16   2           Output(Cmd1Reg)=Mode;          /* cascade timers 3&5, for the
                                                    reeiver$timer$out timer, byte &
                                                    output mode */
 17   2           Call Load_Int_Table;
 18   2           Enable;

 19   2         End Init56;
                /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */


                /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
                /*     Procedure:      Load Interrupt Address Vectors                */
 20   1         Load_Int_Table: Procedure Public;

 21   2           Call Set$Interrupt(42H,EXTINT);
 22   2           Call Set$Interrupt(44H,Receive_Char);
 23   2           Call Set$Interrupt(45H,Transmit_Char);
 24   2           Call Set$Interrupt(46H,Timer_2_4);

 25   2         End Load_Int_Table;
                /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */


                /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
                /*            EXTINT Interrpt Procedure:                            */
                /*                                                                  */
```

231125-10

```
                /*                Service routine reads the Port 1 RS232          */
                /*                handshake signals and sets the message pointer  */
                /*                coresponding to the signal detected.            */
                /*                                                                */
                /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
 26   1         EXTINT: Procedure Interrupt 42H;

 27   2           Enable;
 28   2           HndShk_Pins=Input(Port1Reg) and Port1_Strip;
 29   2           If CTS_Flag = HndShk_Pins and CTS Then
 30   2             Do;
 31   3               Message_Ptr=@CTS_MSG(0);
 32   3               Output(Timer2Reg)=100;
 33   3             End;
 34   2           Else
                    If DSR_Flag = HndShk_Pins and DSR Then
 35   2                Message_Ptr=@DSR_MSG(0);
 36   2             Else
                      If CD_Flag = HndShk_Pins and CD Then
 37   2                  Message_Ptr=@CD_MSG(0);
 38   2               Else
                        If DSRS_Flag = HndShk_Pins and DSRS Then
 39   2                    DO;
 40   3                      Message_Ptr=@DSRS_MSG(0);
 41   3                      If MCI = A Then
 42   3                        Output(Cmd1Reg)=Cmd2A;      /* odd parity, system clk=1.024mhz,
                                                               and 1200 bps */
                              Else
 43   3                        If MCI = B Then
 44   3                          Output(Cmd1Reg)=Cmd2B;    /* odd parity, system clk=1.024mhz,
                                                               and 300 bps */
 45   3                    End;
 46   2           Call Send_Msg;
 47   2           OutPut(RstIntReg)=Int_Reset;

 48   2         End EXTINT;
                /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */


                /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
                /*       Procedure       Receive a character                      */
 49   1         Receive_Char: Procedure Interrupt 44H;

 50   2           Enable;
 51   2           Status=(Input(StatReg) and SioRxRdy);
 52   2           If Status AND Break Then
 53   2             DO;
 54   3               Message_Ptr=@Break_MSG(0);
 55   3               Call Send_Msg;
 56   3             End;
 57   2           Else
                    Do;
 58   3               Char=Input(RxBuffReg) and StripTo7fh;
 59   3               Call MCO(Char);
 60   3             End;
 61   2           OutPut(RstIntReg)=Int_Reset;

 62   2         End Receive_Char;
```

231125-11

```
                /* - - -
```

```
PL/M-86 COMPILER    MAINMOD

  89   1    End MainMod;


MODULE INFORMATION:

     CODE AREA SIZE     = 0235H    565D
     CONSTANT AREA SIZE = 00BEH    190D
     VARIABLE AREA SIZE = 0007H      7D
     MAXIMUM STACK SIZE = 0034H     52D
     280 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS

DICTIONARY SUMMARY:

     31KB MEMORY AVAILABLE
     6KB MEMORY USED   (19%)
     0KB DISK SPACE USED

END OF PL/M-86 COMPILATION
```

intel®

8272A
SINGLE/DOUBLE DENSITY FLOPPY DISK CONTROLLER

- IBM Compatible in Both Single and Double Density Recording Formats
- Programmable Data Record Lengths: 128, 256, 512, or 1024 Bytes/Sector
- Multi-Sector and Multi-Track Transfer Capability
- Drives Up to 4 Floppy or Mini-Floppy Disks
- Data Transfers in DMA or Non-DMA Mode
- Parallel Seek Operations on Up to Four Drives
- Compatible with all Intel and Most Other Microprocessors
- Single-Phase 8 MHz Clock
- Single +5 Volt Power Supply (±10%)

The 8272A is an LSI Floppy Disk Controller (FDC) Chip, which contains the circuitry and control functions for interfacing a processor to 4 Floppy Disk Drives. It is capable of supporting either IBM 3740 single density format (FM), or IBM System 34 Double Density format (MFM) including double sided recording. The 8272A provides control signals which simplify the design of an external phase locked loop and write precompensation circuitry. The FDC simplifies and handles most of the burdens associated with implementing a Floppy Disk Drive Interface. The 8272A is a pin-compatible upgrade to the 8272.

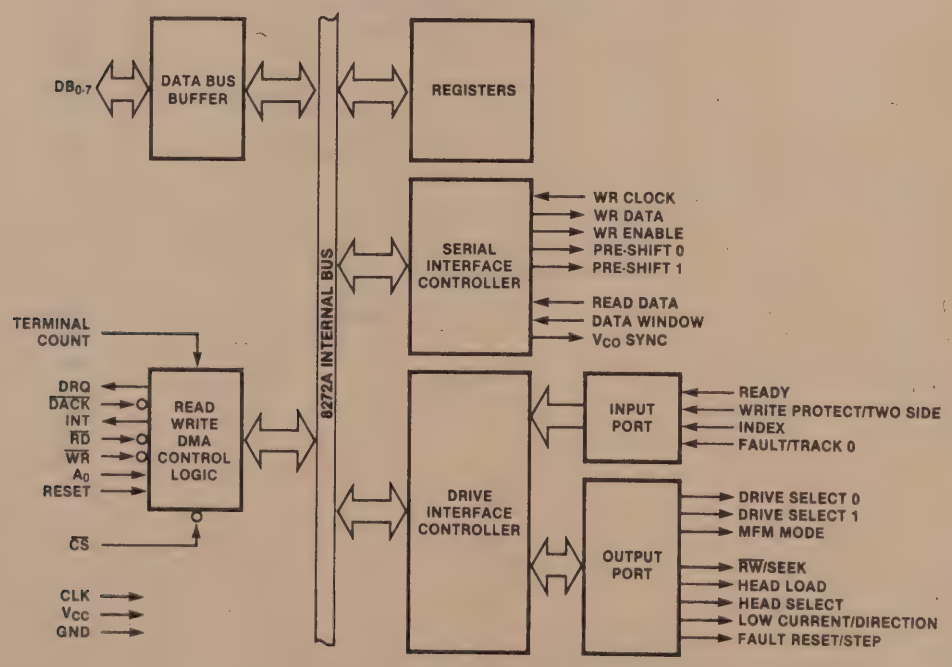


Figure 1. 8272A Internal Block Diagram

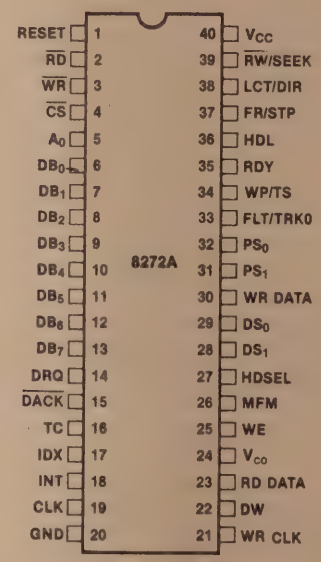


Figure 2. Pin Configuration

Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.
© Intel Corporation, 1982

ORDER NUMBER 210606-001

Table 1. Pin Description

Symbol	Pin No.	Type	Connection To	Name and Function
RESET	1	I	μP	**Reset:** Places FDC in idle state. Resets output lines to FDD to "0" (low). Does not clear the last specify command.
$\overline{RD}$	2	I[1]	μP	**Read:** Control signal for transfer of data from FDC to Data Bus, when "0" (low).
$\overline{WR}$	3	I[1]	μP	**Write:** Control signal for transfer of data to FDC via Data Bus, when "0" (low).
$\overline{CS}$	4	I	μP	**Chip Select:** IC selected when "0" (low), allowing $\overline{RD}$ and $\overline{WR}$ to be enabled.
A_0	5	I[1]	μP	**Data/Status Register Select:** Selects Data Reg (A_0 = 1) or Status Reg (A_0 = 0) contents to be sent to Data Bus.
DB_0-DB_7	6–13	I/O[1]	μP	**Data Bus:** Bidirectional 8-Bit Data Bus.
DRQ	14	O	DMA	**Data DMA Request:** DMA Request is being made by FDC when DRQ "1."[3]
$\overline{DACK}$	15	I	DMA	**DMA Acknowledge:** DMA cycle is active when "0" (low) and Controller is performing DMA transfer.
TC	16	I	DMA	**Terminal Count:** Indicates the termination of a DMA transfer when "1" (high)[2].
IDX	17	I	FDD	**Index:** Indicates the beginning of a disk track.
INT	18	O	μP	**Interrupt:** Interrupt Request Generated by FDC.
CLK	19	I		**Clock:** Single Phase 8 MHz (4 MHz for mini floppies) Squarewave Clock.
GND	20			**Ground:** D.C. Power Return.

Note 1: Disabled when $\overline{CS}$=1.
Note 2: TC must be activated to terminate the Execution Phase of any command.
Note 3: DRQ is also an input for certain test modes. It should have a 5kΩ pull-up resistor to prevent activation.

Symbol	Pin No.	Type	Connection To	Name and Function
V_{CC}	40			**D.C. Power:** +5V
$\overline{RW}$/SEEK	39	O	FDD	**Read Write / SEEK:** When "1" (high) Seek mode selected and when "0" (low) Read/Write mode selected.
LCT/DIR	38	O	FDD	**Low Current/Direction:** Lowers Write current on inner tracks in Read/Write mode, determines direction head will step in Seek mode.
FR/STP	37	O	FDD	**Fault Reset/Step:** Resets fault FF in FDD in Read/Write mode, provides step pulses to move head to another cylinder in Seek mode.
HDL	36	O	FDD	**Head Load:** Command which causes read/write head in FDD to contact diskette.
RDY	35	I	FDD	**Ready:** Indicates FDD is ready to send or receive data. Must be tied high (gated by the index pulse) for mini floppies which do not normally have a Ready line.
WP/TS	34	I	FDD	**Write Protect / Two-Side:** Senses Write Protect status in Read/Write mode, and Two Side Media in Seek mode.
FLT/TRK0	33	I	FDD	**Fault/Track 0:** Senses FDD fault condition in Read/Write mode and Track 0 condition in Seek mode.
PS_1,PS_0	31,32	O	FDD	**Precompensation (preshift):** Write precompensation status during MFM mode. Determines early, late, and normal times.
WR DATA	30	O	FDD	**Write Data:** Serial clock and data bits to FDD.
DS_1,DS_0	28,29	O	FDD	**Drive Select:** Selects FDD unit.
HDSEL	27	O	FDD	**Head Select:** Head 1 selected when "1" (high) Head 0 selected when "0" (low).

Table 1. Pin Description (Continued)

Symbol	Pin No.	Type	Connection To	Name and Function
MFM	26	O	PLL	**MFM Mode:** MFM mode when "1," FM mode when "0."
WE	25	O	FDD	**Write Enable:** Enables write data into FDD.
VCO	24	O	PLL	**VCO Sync:** Inhibits VCO in PLL when "0" (low), enables VCO when "1."
RD DATA	23	I	FDD	**Read Data:** Read data from FDD, containing clock and data bits.

Symbol	Pin No.	Type	Connection To	Name and Function
DW	22	I	PLL	**Data Window:** Generated by PLL, and used to sample data from FDD.
WR CLK	21	I		**Write Clock:** Write data rate to FDD FM = 500 kHz, MFM = 1 MHz, with a pulse width of 250 ns for both FM and MFM. Must be enabled for all operations, both Read and Write.

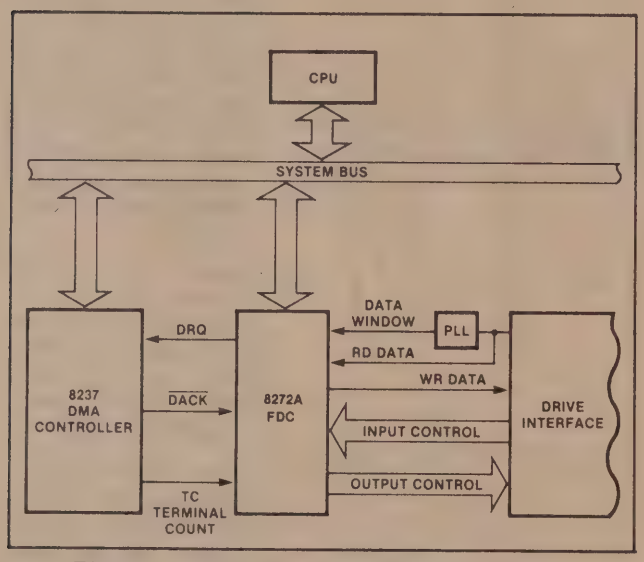


Figure 3. 8272A System Block Diagram

DESCRIPTION

Hand-shaking signals are provided in the 8272A which make DMA operation easy to incorporate with the aid of an external DMA Controller chip, such as the 8237A. The FDC will operate in either DMA or Non-DMA mode. In the Non-DMA mode, the FDC generates interrupts to the processor for every transfer of a data byte between the CPU and the 8272A. In the DMA mode, the processor need only load a command into the FDC and all data transfers occur under control of the 8272A and DMA controller.

There are 15 separate commands which the 8272A will execute. Each of these commands require multiple 8-bit bytes to fully specify the operation which the processor wishes the FDC to perform. The following commands are available.

- Read Data
- Read ID
- Read Deleted Data
- Read a Track
- Scan Equal
- Scan High or Equal
- Scan Low or Equal
- Specify
- Write Data
- Format a Track
- Write Deleted Data
- Seek
- Recalibrate (Restore to Track 0)
- Sense Interrupt Status
- Sense Drive Status

For more information see the Intel Application Notes AP-116 and AP-121.

FEATURES

Address mark detection circuitry is internal to the FDC which simplifies the phase locked loop and read electronics. The track stepping rate, head load time, and head unload time may be programmed by the user. The 8272A offers many additional features such as multiple sector transfers in both read and write modes with a single command, and full IBM compatibility in both single (FM) and double density (MFM) modes.

8272A ENHANCEMENTS

On the 8272A, after detecting the Index Pulse, the VCO Sync output stays low for a shorter period of time. See Figure 4A.

On the 8272 there can be a problem reading data when Gap 4A is 00 and there is no IAM. This occurs on some older floppy formats. The 8272A cures this problem by adjusting the VCO Sync timing so that it is not low during the data field. See Figure 4B.

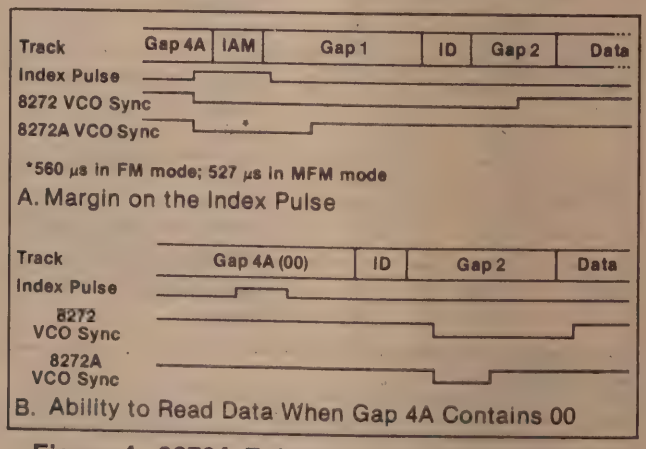


Figure 4. 8272A Enhancements over the 8272

8272A REGISTERS — CPU INTERFACE

The 8272A contains two registers which may be accessed by the main system processor; a Status Register and a Data Register. The 8-bit Main Status Register contains the status information of the FDC, and may be accessed at any time. The 8-bit Data Register (actually consists of several registers in a stack with only one register presented to the data bus at a time), stores data, commands, parameters, and FDD status information. Data bytes are read out of, or written into, the Data Register in order to program or obtain the results after execution of a command. The Status Register may only be read and is used to facilitate the transfer of data between the processor and 8272A.

The relationship between the Status/Data registers and the signals $\overline{RD}$, $\overline{WR}$, and A_0 is shown in Table 2.

Table 2. A_0, $\overline{RD}$, $\overline{WR}$ decoding for the selection of Status/Data register functions.

A_0	$\overline{RD}$	$\overline{WR}$	FUNCTION
0	0	1	Read Main Status Register
0	1	0	Illegal (see note)
0	0	0	Illegal (see note)
1	0	0	Illegal (see note)
1	0	1	Read from Data Register
1	1	0	Write into Data Register

Note: Design must guarantee that the 8272A is not subjected to illegal inputs.

The Main Status Register bits are defined in Table 3.

Table 3. Main Status Register bit description.

BIT NUMBER	NAME	SYMBOL	DESCRIPTION
D_0	FDD 0 Busy	D_0B	FDD number 0 is in the Seek mode.
D_1	FDD 1 Busy	D_1B	FDD number 1 is in the Seek mode.
D_2	FDD 2 Busy	D_2B	FDD number 2 is in the Seek mode.
D_3	FDD 3 Busy	D_3B	FDD number 3 is in the Seek mode.
D_4	FDC Busy	CB	A read or write command is in process.
D_5	Non-DMA mode	NDM	The FDC is in the non-DMA mode. This bit is set only during the execution phase in non-DMA mode. Transition to "0" state indicates execution phase has ended.
D_6	Data Input/Output	DIO	Indicates direction of data transfer between FDC and Dta Register. If DIO = "1" then transfer is from Data Register to the Processor. If DIO = "0", then transfer is from the Processor to Data Register.
D_7	Request for Master	RQM	Indicates Data Register is ready to send or receive data to or from the Processor. Both bits DIO and RQM should be used to perform the handshaking functions of "ready" and "direction" to the processor.

The DIO and RQM bits in the Status Register indicate when Data is ready and in which direction data will be transferred on the Data Bus.

Note: There is a 12μS or 24μS RQM flag delay when using an 8 or 4 MHz clock respectively.

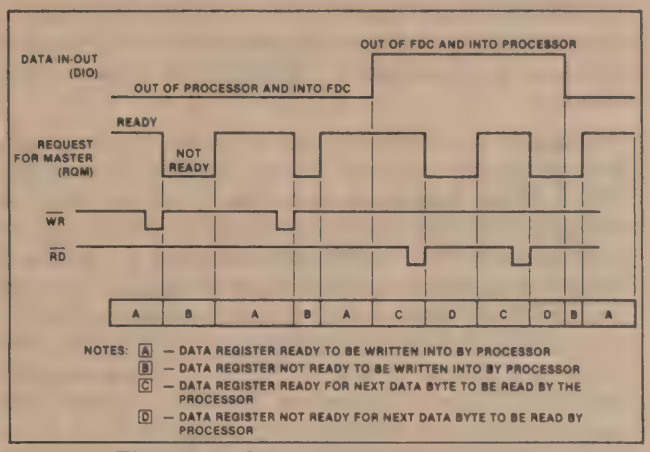


Figure 5. Status Register Timing

The 8272A is capable of executing 15 different commands. Each command is initiated by a multi-byte transfer from the processor, and the result after execution of the command may also be a multi-byte transfer back to the processor. Because of this multi-byte interchange of information between the 8272A and the processor, it is convenient to consider each command as consisting of three phases:

Command Phase: The FDC receives all information required to perform a particular operation from the processor.

Execution Phase: The FDC performs the operation it was instructed to do.

Result Phase: After completion of the operation, status and other housekeeping information are made available to the processor.

During Command or Result Phases the Main Status Register (described in Table 3) must be read by the processor before each byte of information is written into or read from the Data Register. Bits D6 and D7 in the Main Status Register must be in a 0 and 1 state, respectively, before each byte of the command word may be written into the 8272A. Many of the commands require multiple bytes, and as a result the Main Status Register must be read prior to each byte transfer to the 8272A. On the other hand, during the Result Phase, D6 and D7 in the Main Status Register must both be 1's (D6 = 1 and D7 = 1) before reading each byte from the Data Register. Note, this reading of the Main Status Register before each byte transfer to the 8272A is required in only the Command and Result Phases, and NOT during the Execution Phase.

During the Execution Phase, the Main Status Register need not be read. If the 8272A is in the non-DMA Mode, then the receipt of each data byte (if 8272A is reading data from FDD) is indicated by an Interrupt signal on pin 18 (INT = 1). The generation of a Read signal (RD = 0) will reset the Interrupt as well as output the Data onto

the Data Bus. For example, if the processor cannot handle Interrupts fast enough (every 13 μs for MFM mode) then it may poll the Main Status Register and then bit D7 (RQM) functions just like the Interrupt signal. If a Write Command is in process, then the WR signal performs the reset to the Interrupt signal.

The 8272A always operates in a multi-sector transfer mode. It continues to transfer data until the TC input is active. In Non-DMA Mode, the system must supply the TC input.

If the 8272A is in the DMA Mode, no Interrupts are generated during the Execution Phase. The 8272A generates DRQ's (DMA Requests) when each byte of data is available. The DMA Controller responds to this request with both a $\overline{DACK}=0$ (DMA Acknowledge) and a $\overline{RD}=0$ (Read signal). When the DMA Acknowledge signal goes low ($\overline{DACK}=0$) then the DMA Request is reset (DRQ = 0). If a Write Command has been programmed then a $\overline{WR}$ signal will appear instead of $\overline{RD}$. After the Execution Phase has been completed (Terminal Count has occurred) then an Interrupt will occur (INT = 1). This signifies the beginning of the Result Phase. When the first byte of data is read during the Result Phase, the Interrupt is automatically reset (INT = 0).

It is important to note that during the Result Phase all bytes shown in the Command Table must be read. The Read Data Command, for example, has seven bytes of data in the Result Phase. All seven bytes must be read in order to successfully complete the Read Data Command. The 8272A will not accept a new command until all seven bytes have been read. Other commands may require fewer bytes to be read during the Result Phase.

The 8272A contains five Status Registers. The Main Status Register mentioned above may be read by the processor at any time. The other four Status Registers (ST0, ST1, ST2, and ST3) are only available during the Result Phase, and may be read only after successfully completing a command. The particular command which has been executed determines how many of the Status Registers will be read.

The bytes of data which are sent to the 8272A to form the Command Phase, and are read out of the 8272A in the Result Phase, must occur in the order shown in the Table 4. That is, the Command Code must be sent first and the other bytes sent in the prescribed sequence. No foreshortening of the Command or Result Phases are allowed. After the last byte of data in the Command Phase is sent to the 8272A, the Execution Phase

Table 4. 8272A Command Set

PHASE	R/W	DATA BUS D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	REMARKS
READ DATA										
Command	W	MT	MFM	SK	0	0	1	1	0	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W	C								Sector ID information prior to Command execution
	W	H								
	W	R								
	W	N								
	W	EOT								
	W	GPL								
	W	DTL								
Execution										Data transfer between the FDD and main-system
Result	R	ST 0								Status information after Command execution
	R	ST 1								
	R	ST 2								
	R	C								Sector ID information after command execution
	R	H								
	R	R								
	R	N								
READ DELETED DATA										
Command	W	MT	MFM	SK	0	1	1	0	0	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W	C								Sector ID information prior to Command execution
	W	H								
	W	R								
	W	N								
	W	EOT								
	W	GPL								
	W	DTL								
Execution										Data transfer between the FDD and main-system
Result	R	ST 0								Status information after Command execution
	R	ST 1								
	R	ST 2								
	R	C								Sector ID information after Command execution
	R	H								
	R	R								
	R	N								

PHASE	R/W	DATA BUS D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	REMARKS
WRITE DATA										
Command	W	MT	MFM	0	0	0	1	0	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W	C								Sector ID information prior to Command execution
	W	H								
	W	R								
	W	N								
	W	EOT								
	W	GPL								
	W	DTL								
Execution										Data transfer between the main-system and FDD
Result	R	ST 0								Status information after Command execution
	R	ST 1								
	R	ST 2								
	R	C								Sector ID information after Command execution
	R	H								
	R	R								
	R	N								
WRITE DELETED DATA										
Command	W	MT	MFM	0	0	1	0	0	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W	C								Sector ID information prior to Command execution
	W	H								
	W	R								
	W	N								
	W	EOT								
	W	GPL								
	W	DTL								
Execution										Data transfer between the FDD and main-system
Result	R	ST 0								Status information after Command execution
	R	ST 1								
	R	ST 2								
	R	C								Sector ID information after Command execution
	R	H								
	R	R								
	R	N								

Note: 1. Symbols used in this table are described at the end of this section.
2. $A_0 = 1$ for all operations.
3. X = Don't care, usually made to equal binary 0.

Table 4. 8272A Command Set (Continued)

PHASE	R/W	DATA BUS D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	REMARKS
READ A TRACK										
Command	W	0	MFM	SK	0	0	0	1	0	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W	C								Sector ID information prior to Command execution
	W	H								
	W	R								
	W	N								
	W	EOT								
	W	GPL								
	W	DTL								
Execution										Data transfer between the FDD and main-system. FDC reads all of cylinders contents from index hole to EOT
Result	R	ST 0								Status information after Command execution
	R	ST 1								
	R	ST 2								
	R	C								Sector ID information after Command execution
	R	H								
	R	R								
	R	N								
READ ID										
Command	W	0	MFM	0	0	1	0	1	0	Commands
	W	0	0	0	0	0	HDS	DS1	DS0	
Execution										The first correct ID information on the Cylinder is stored in Data Register
Result	R	ST 0								Status information after Command execution
	R	ST 1								
	R	ST 2								
	R	C								Sector ID information during Execution Phase
	R	H								
	R	R								
	R	N								
FORMAT A TRACK										
Command	W	0	MFM	0	0	1	1	0	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W	N								Bytes/Sector
	W	SC								Sectors/Cylinder
	W	GPL								Gap 3
	W	D								Filler Byte
Execution										FDC formats an entire cylinder
Result	R	ST 0								Status information after Command execution
	R	ST 1								
	R	ST 2								
	R	C								In this case, the ID information has no meaning
	R	H								
	R	R								
	R	N								
SCAN EQUAL										
Command	W	MT	MFM	SK	1	0	0	0	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W	C								Sector ID information prior to Command execution
	W	H								
	W	R								
	W	N								
	W	EOT								
	W	GPL								
	W	STP								
Execution										Data compared between the FDD and main-system
Result	R	ST 0								Status information after Command execution
	R	ST 1								
	R	ST 2								
	R	C								Sector ID information after Command execution
	R	H								
	R	R								
	R	N								
SCAN LOW OR EQUAL										
Command	W	MT	MFM	SK	1	1	0	0	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W	C								Sector ID information prior Command execution
	W	H								
	W	R								
	W	N								
	W	EOT								
	W	GPL								
	W	STP								
Execution										Data compared between the FDD and main-system
Result	R	ST 0								Status information after Command execution
	R	ST 1								
	R	ST 2								
	R	C								Sector ID information after Command execution
	R	H								
	R	R								
	R	N								
SCAN HIGH OR EQUAL										
Command	W	MT	MFM	SK	1	1	1	0	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W	C								Sector ID information prior Command execution
	W	H								
	W	R								
	W	N								
	W	EOT								
	W	GPL								
	W	STP								
Execution										Data compared between the FDD and main-system
Result	R	ST 0								Status information after Command execution
	R	ST 1								
	R	ST 2								
	R	C								Sector ID information after Command execution
	R	H								
	R	R								
	R	N								
RECALIBRATE										
Command	W	0	0	0	0	0	1	1	1	Command Codes
	W	0	0	0	0	0	0	DS1	DS0	
Execution										Head retracted to Track 0
SENSE INTERRUPT STATUS										
Command	W	0	0	0	0	1	0	0	0	Command Codes
Result	R	ST 0								Status information at the end of each seek operation about the FDC
	R	PCN								
SPECIFY										
Command	W	0	0	0	0	0	0	1	1	Command Codes
	W	SRT						HUT		
	W	HLT						ND		
SENSE DRIVE STATUS										
Command	W	0	0	0	0	0	1	0	0	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
Result	R	ST 3								Status information about FDD
SEEK										
Command	W	0	0	0	0	1	1	1	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W	NCN								
Execution										Head is positioned over proper Cylinder on Diskette
INVALID										
Command	W	Invalid Codes								Invalid Command Codes (NoOp — FDC goes into Standby State)
Result	R	ST 0								ST 0 = 80 (16)

Table 5. Command Mneumonics

SYMBOL	NAME	DESCRIPTION
A_0	Address Line 0	A_0 controls selection of Main Status Register ($A_0=0$) or Data Register ($A_0=1$).
C	Cylinder Number	C stands for the current selected Cylinder track number 0 through 76 of the medium.
D	Data	D stands for the data pattern which is going to be written into a Sector.
D_7-D_0	Data Bus	8-bit Data Bus where D_7 is the most significant bit, and D_0 is the least significant bit.
DS0, DS1	Drive Select	DS stands for a selected drive number 0 or 1.
DTL	Data Length	When N is defined as 00, DTL stands for the data length which users are going to read out or write into the Sector.
EOT	End of Track	EOT stands for the final Sector number of a Cylinder.
GPL	Gap Length	GPL stands for the length of Gap 3 (spacing between Sectors excluding VCO Sync Field).
H	Head Address	H stands for head number 0 or 1, as specified in ID field.
HDS	Head Select	HDS stands for a selected head number 0 or 1 (H = HDS in all command words).
HLT	Head Load Time	HLT stands for the head load time in the FDD (2 to 254 ms in 2 ms increments).
HUT	Head Unload Time	HUT stands for the head unload time after a read or write operation has occurred (16 to 240 ms in 16 ms increments).
MFM	FM or MFM Mode	If MF is low, FM mode is selected and if it is high, MFM mode is selected.
MT	Multi-Track	If MT is high, a multi-track operation is to be performed (a cylinder under both HD0 and HD1 will be read or written).
N	Number	N stands for the number of data bytes written in a Sector.
NCN	New Cylinder Number	NCN stands for a new Cylinder number, which is going to be reached as a result of the Seek operation. Desired position of Head.
ND	Non-DMA Mode	ND stands for operation in the Non-DMA Mode.
PCN	Present Cylinder Number	PCN stands for the Cylinder number at the completion of SENSE INTERRUPT STATUS Command. Position of Head at present time.
R	Record	R stands for the Sector number, which will be read or written.
R/W	Read/Write	R/W stands for either Read (R) or Write (W) signal.
SC	Sector	SC indicates the number of Sectors per Cylinder.
SK	Skip	SK stands for Skip Deleted Data Address Mark.
SRT	Step Rate Time	SRT stands for the Stepping Rate for the FDD (1 to 16 ms in 1 ms increments). The same Stepping Rate applies to all drives (F=1 ms, E=2 ms, etc.).
ST 0 ST 1 ST 2 ST 3	Status 0 Status 1 Status 2 Status 3	ST 0-3 stand for one of four registers which store the status information after a command has been executed. This information is available during the result phase after command execution. These registers should not be confused with the main status register (selected by $A_0=0$). ST 0-3 may be read only after a command has been executed and contain information relevant to that particular command.
STP		During a Scan operation, if STP = 1, the data in contiguous sectors is compared byte by byte with data sent from the processor (or DMA), and if STP = 2, then alternate sectors are read and compared.

automatically starts. In a similar fashion, when the last byte of data is read out in the Result Phase, the command is automatically ended and the 8272A is ready for a new command. A command may be aborted by simply sending a Terminal Count signal to pin 16 (TC = 1). This is a convenient means of ensuring that the processor may always get the 8272A's attention even if the disk system hangs up in an abnormal manner.

POLLING FEATURE OF THE 8272A

After power-up RESET, the Drive Select Lines DS0 and DS1 will automatically go into a polling mode. In between commands (and between step pulses in the SEEK command) the 8272A polls all four FDDs looking for a change in the Ready line from any of the drives. If the Ready line changes state (usually due to a door opening or closing) then the 8272A will generate an interrupt. When Status Register 0 (ST0) is read (after Sense Interrupt Status is issued), Not Ready (NR) will be indicated. The polling of the Ready line by the 8272A occurs continuously between instructions, thus notifying the processor which drives are on or off line. Approximate scan timing is shown in Table 6.

Table 6. Scan Timing

DS1	DS0	APPROXIMATE SCAN TIMING
0	0	220μS
0	1	220μS
1	0	220μS
1	1	440μS

COMMAND DESCRIPTIONS

During the Command Phase, the Main Status Register must be polled by the CPU before each byte is written into the Data Register. The DIO (DB6) and RQM (DB7) bits in the Main Status Register must be in the "0" and "1" states respectively, before each byte of the command may be written into the 8272A. The beginning of the execution phase for any of these commands will cause DIO and RQM to switch to "1" and "0" states respectively.

READ DATA

A set of nine (9) byte words are required to place the FDC into the Read Data Mode. After the Read Data command has been issued the FDC loads the head (if it is in the unloaded state), waits the specified head settling time (defined in the Specify Command), and begins reading ID Address Marks and ID fields. When the current sector number ("R") stored in the ID Register (IDR) compares with the sector number read off the diskette, then the FDC outputs data (from the data field) byte-by-byte to the main system via the data bus.

After completion of the read operation from the current sector, the Sector Number is incremented by one, and the data from the next sector is read and output on the data bus. This continuous read function is called a "Multi-Sector Read Operation." The Read Data Command must be terminated by the receipt of a Terminal Count signal. Upon receipt of this signal, the FDC stops outputting data to the processor, but will continue to read data from the current sector, check CRC (Cyclic Redundancy Count) bytes, and then at the end of the sector terminate the Read Data Command.

The amount of data which can be handled with a single command to the FDC depends upon MT (multi-track), MFM (MFM/FM), and N (Number of Bytes/Sector). Table 7 on the next page shows the Transfer Capacity.

Table 7. Transfer Capacity

Multi-Track MT	MFM/FM MFM	Bytes/Sector N	Maximum Transfer Capacity (Bytes/Sector) (Number of Sectors)	Final Sector Read from Diskette
0 0	0 1	00 01	(128) (26) = 3,328 (256) (26) = 6,656	26 at Side 0 or 26 at Side 1
1 1	0 1	00 01	(128) (52) = 6,656 (256) (52) = 13,312	26 at Side 1
0 0	0 1	01 02	(256) (15) = 3,840 (512) (15) = 7,680	15 at Side 0 or 15 at Side 1
1 1	0 1	01 02	(256) (30) = 7,680 (512) (30) = 15,360	15 at Side 1
0 0	0 1	02 03	(512) (8) = 4,096 (1024) (8) = 8,192	8 at Side 0 or 8 at Side 1
1 1	0 1	02 03	(512) (16) = 8,192 (1024) (16) = 16,384	8 at Side 1

The "multi-track" function (MT) allows the FDC to read data from both sides of the diskette. For a particular cylinder, data will be transferred starting at Sector 1, Side 0 and completing at Sector L, Side 1 (Sector L = last sector on the side). Note, this function pertains to only one cylinder (the same track) on each side of the diskette.

When N = 0, then DTL defines the data length which the FDC must treat as a sector. If DTL is smaller than the actual data length in a Sector, the data beyond DTL in the Sector is not sent to the Data Bus. The FDC reads (internally) the complete Sector performing the CRC check, and depending upon the manner of command termination, may perform a Multi-Sector Read Operation. When N is non-zero, then DTL has no meaning and should be set to 0FFH.

At the completion of the Read Data Command, the head is not unloaded until after Head Unload Time Interval (specified in the Specify Command) has elapsed. If the processor issues another command before the head unloads then the head settling time may be saved between subsequent reads. This time out is particularly valuable when a diskette is copied from one drive to another.

If the FDC detects the Index Hole twice without finding the right sector, (indicated in "R"), then the FDC sets the ND (No Data) flag in Status Register 1 to a 1 (high), and terminates the Read Data Command. (Status Register 0 also has bits 7 and 6 set to 0 and 1 respectively.)

After reading the ID and Data Fields in each sector, the FDC checks the CRC bytes. If a read error is detected (incorrect CRC in ID field), the FDC sets the DE (Data Error) flag in Status Register 1 to a 1 (high), and if a CRC error occurs in the Data Field the FDC also sets the DD (Data Error in Data Field) flag in Status Register 2 to a 1 (high), and terminates the Read Data Command. (Status Register 0 also has bits 7 and 6 set to 0 and 1 respectively.)

If the FDC reads a Deleted Data Address Mark off the diskette, and the SK bit (bit D5 in the first Command Word) is not set (SK = 0), then the FDC sets the CM (Control Mark) flag in Status Register 2 to a 1 (high), and terminates the Read Data Command, after reading all the data in the Sector. If SK = 1, the FDC skips the sector with the Deleted Data Address Mark and reads the next sector.

During disk data transfers between the FDC and the processor, via the data bus, the FDC must be serviced by the processor every 27 μs in the FM Mode, and every 13 μs in the MFM Mode, or the FDC sets the OR (Over Run) flag in Status Register 1 to a 1 (high), and terminates the Read Data Command.

If the processor terminates a read (or write) operation in the FDC, then the ID Information in the Result Phase is dependent upon the state of the MT bit and EOT byte. Table 5 shows the values for C, H, R, and N, when the processor terminates the Command.

Table 8. ID Information When Processor Terminates Command

MT	EOT	Final Sector Transferred to Processor	ID Information at Result Phase			
			C	H	R	N
0	1A 0F 08	Sector 1 to 25 at Side 0 Sector 1 to 14 at Side 0 Sector 1 to 7 at Side 0	NC	NC	R + 1	NC
	1A 0F 08	Sector 26 at Side 0 Sector 15 at Side 0 Sector 8 at Side 0	C + 1	NC	R = 01	NC
	1A 0F 08	Sector 1 to 25 at Side 1 Sector 1 to 14 at Side 1 Sector 1 to 7 at Side 1	NC	NC	R + 1	NC
	1A 0F 08	Sector 26 at Side 1 Sector 15 at Side 1 Sector 8 at Side 1	C + 1	NC	R = 01	NC
1	1A 0F 08	Sector 1 to 25 at Side 0 Sector 1 to 14 at Side 0 Sector 1 to 7 at Side 0	NC	NC	R + 1	NC
	1A 0F 08	Sector 26 at Side 0 Sector 15 at Side 0 Sector 8 at Side 0	NC	LSB	R = 01	NC
	1A 0F 08	Sector 1 to 25 at Side 1 Sector 1 to 14 at Side 1 Sector 1 to 7 at Side 1	NC	NC	R + 1	NC
	1A 0F 08	Sector 26 at Side 1 Sector 15 at Side 1 Sector 8 at Side 1	C + 1	LSB	R = 01	NC

Notes: 1. NC (No Change): The same value as the one at the beginning of command execution.
2. LSB (Least Significant Bit): The least significant bit of H is complemented.

WRITE DATA

A set of nine (9) bytes are required to set the FDC into the Write Data mode. After the Write Data command has been issued the FDC loads the head (if it is in the unloaded state), waits the specified head settling time (defined in the Specify Command), and begins reading ID Fields. When the current sector number ("R"), stored in the ID Register (IDR) compares with the sector

number read off the diskette, then the FDC takes data from the processor byte-by-byte via the data bus, and outputs it to the FDD.

After writing data into the current sector, the Sector Number stored in "R" is incremented by one, and the next data field is written into. The FDC continues this "Multi-Sector Write Operation" until the issuance of a Terminal Count signal. If a Terminal Count signal is sent to the FDC it continues writing into the current sector to complete the data field. If the Terminal Count signal is received while a data field is being written then the remainder of the data field is filled with 00 (zeros).

The FDC reads the ID field of each sector and checks the CRC bytes. If the FDC detects a read error (incorrect CRC) in one of the ID Fields, it sets the DE (Data Error) flag of Status Register 1 to a 1 (high), and terminates the Write Data Command. (Status Register 0 also has bits 7 and 6 set to 0 and 1 respectively.)

The Write Command operates in much the same manner as the Read Command. The following items are the same; refer to the Read Data Command for details:

- Transfer Capacity
- EN (End of Cylinder) Flag
- ND (No Data) Flag
- Head Unload Time Interval
- ID Information when the processor terminates command (see Table 2)
- Definition of DTL when N = 0 and when N ≠ 0

In the Write Data mode, data transfers between the processor and FDC must occur every 31 μs in the FM mode, and every 15 μs in the MFM mode. If the time interval between data transfers is longer than this then the FDC sets the OR (Over Run) flag in Status Register 1 to a 1 (high), and terminates the Write Data Command.

For mini-floppies, multiple track writes are usually not permitted. This is because of the turn-off time of the erase head coils—the head switches tracks before the erase head turns off. Therefore the system should typically wait 1.3 mS before attempting to step or change sides.

WRITE DELETED DATA

This command is the same as the Write Data Command except a Deleted Data Address Mark is written at the beginning of the Data Field instead of the normal Data Address Mark.

READ DELETED DATA

This command is the same as the Read Data Command except that when the FDC detects a Data Address Mark at the beginning of a Data Field (and SK = 0 (low)), it will read all the data in the sector and set the CM flag in Status Register 2 to a 1 (high), and then terminate the command. If SK = 1, then the FDC skips the sector with the Data Address Mark and reads the next sector.

READ A TRACK

This command is similar to READ DATA Command except that the entire data field is read continuously from each of the sectors of a track. Immediately after encountering the INDEX HOLE, the FDC starts reading all data fields on the track as continuous blocks of data. If the FDC finds an error in the ID or DATA CRC check bytes, it continues to read data from the track. The FDC compares the ID information read from each sector with the value stored in the IDR, and sets the ND flag of Status Register 1 to a 1 (high) if there is no comparison. Multi-track or skip operations are not allowed with this command.

This command terminates when EOT number of sectors have been read. If the FDC does not find an ID Address Mark on the diskette after it encounters the INDEX HOLE for the second time, then it sets the MA (missing address mark) flag in Status Register 1 to a 1 (high), and terminates the command. (Status Register 0 has bits 7 and 6 set to 0 and 1 respectively.)

READ ID

The READ ID Command is used to give the present position of the recording head. The FDC stores the values from the first ID Field it is able to read. If no proper ID Address Mark is found on the diskette, before the INDEX HOLE is encountered for the second time then the MA (Missing Address Mark) flag in Status Register 1 is set to a 1 (high), and if no data is found then the ND (No Data) flag is also set in Status Register 1 to a 1 (high) and the command is terminated.

FORMAT A TRACK

The Format Command allows an entire track to be formatted. After the INDEX HOLE is detected, Data is written on the Diskette: Gaps, Address Marks, ID Fields and Data Fields, all per the IBM System 34 (Double Density) or System 3740 (Single Density) Format are recorded. The particular format which will be written is controlled by the values programmed into N (number of bytes/sector), SC (sectors/cylinder), GPL (Gap Length), and D (Data Pattern) which are supplied by the processor during the Command Phase. The Data Field is filled with the Byte of data stored in D. The ID Field for each sector is supplied by the processor; that is, four data requests per sector are made by the FDC for C (Cylinder Number), H (Head Number), R (Sector Number) and N (Number of Bytes/Sector). This allows the diskette to be formatted with nonsequential sector numbers, if desired.

After formatting each sector, the processor must send new values for C, H, R, and N to the 8272A for each sector on the track. The contents of the R Register is incremented by one after each sector is formatted, thus, the R register contains a value of R + 1 when it is read during the Result Phase. This incrementing and formatting continues for the whole track until the FDC encounters the INDEX HOLE for the second time, whereupon it terminates the command.

If a FAULT signal is received from the FDD at the end of a write operation, then the FDC sets the EC flag of Status Register 0 to a 1 (high), and terminates the command after setting bits 7 and 6 of Status Register 0 to 0 and 1 respectively. Also the loss of a READY signal at the beginning of a command execution phase causes command termination.

Table 9 shows the relationship between N, SC, and GPL for various sector sizes:

Table 9. Sector Size Relationships.

FORMAT	8" STANDARD FLOPPY SECTOR SIZE	N	SC	GPL[1]	GPL[2]	REMARKS	5¼" MINI FLOPPY SECTOR SIZE	N	SC	GPL[1]	GPL[2]
FM Mode	128 bytes/Sector	00	1A	07	1B	IBM Diskette 1	128 bytes/Sector	00	12	07	09
	256	01	0F	0E	2A	IBM Diskette 2	128	00	10	10	19
	512	02	08	1B	3A		256	01	08	18	30
	1024	03	04	47	8A		512	02	04	46	87
	2048	04	02	C8	FF		1024	03	02	C8	FF
	4096	05	01	C8	FF		2048	04	01	C8	FF
MPM Mode	256	01	1A	0E	36	IBM Diskette 2D	256	01	12	0A	0C
	512	02	OF	1B	54		256	01	10	20	32
	1024	03	08	35	74	IBM Diskette 2D	512	02	08	2A	50
	2048	04	04	99	FF		1024	03	04	80	F0
	4096	05	02	C8	FF		2048	04	02	C8	FF
	8192	06	01	C8	FF		4096	05	01	C8	FF

Note: 1. Suggested values of GPL in Read or Write Commands to avoid splice point between data field and ID field of contiguous sections.

2. Suggested values of GPL in format command.

SCAN COMMANDS

The SCAN Commands allow data which is being read from the diskette to be compared against data which is being supplied from the main system (Processor in NON-DMA mode, and DMA Controller in DMA mode). The FDC compares the data on a byte-by-byte basis, and looks for a sector of data which meets the conditions of $D_{FDD} = D_{Processor}$, $D_{FDD} < D_{Processor}$, or $D_{FDD} \geq D_{Processor}$. Ones complement arithmetic is used for comparison (FF = largest number, 00 = smallest number). After a whole sector of data is compared, if the conditions are not met, the sector number is incremented (R + STP → R), and the scan operation is continued. The scan operation continues until one of the following conditions occur; the conditions for scan are met (equal, low, or high), the last sector on the track is reached (EOT), or the terminal count signal is received.

If the conditions for scan are met then the FDC sets the SH (Scan Hit) flag of Status Register 2 to a 1 (high), and terminates the Scan Command. If the conditions for scan are not met between the starting sector (as specified by R) and the last sector on the cylinder (EOT), then the FDC sets the SN (Scan Not Satisfied) flag of Status Register 2 to a 1 (high), and terminates the Scan Command. The receipt of a TERMINAL COUNT signal from the Processor or DMA Controller during the scan operation will cause the FDC to complete the comparison of the particular byte which is in process, and then to terminate the command. Table 10 shows the status of bits SH and SN under various conditions of SCAN.

Table 10. Scan Status Codes

COMMAND	STATUS REGISTER 2 BIT 2 = SN	STATUS REGISTER 2 BIT 3 = SH	COMMENTS
Scan Equal	0	1	$D_{FDD} = D_{Processor}$
	1	0	$D_{FDD} \neq D_{Processor}$
Scan Low or Equal	0	1	$D_{FDD} = D_{Processor}$
	0	0	$D_{FDD} < D_{Processor}$
	1	0	$D_{FDD} \nless D_{Processor}$
Scan High or Equal	0	1	$D_{FDD} = D_{Processor}$
	0	0	$D_{FDD} > D_{Processor}$
	1	0	$D_{FDD} \ngtr D_{Processor}$

If the FDC encounters a Deleted Data Address Mark on one of the sectors (and SK = 0), then it regards the sector as the last sector on the cylinder, sets CM (Control Mark) flag of Status Register 2 to a 1 (high) and terminates the command. If SK = 1, the FDC skips the sector with the Deleted Address Mark, and reads the next sector. In the second case (SK = 1), the FDC sets the CM (Control Mark) flag of Status Register 2 to a 1 (high) in order to show that a Deleted Sector had been encountered.

When either the STP (contiguous sectors STP = 01, or alternate sectors STP = 02 sectors are read) or the MT (Multi-Track) are programmed, it is necessary to remember that the last sector on the track must be read. For example, if STP = 02, MT = 0, the sectors are numbered sequentially 1 through 26, and we start the Scan Command at sector 21; the following will happen. Sectors 21, 23, and 25 will be read, then the next sector (26) will be skipped and the Index Hole will be encountered before the EOT value of 26 can be read. This will result in an abnormal termination of the command. If the EOT had been set at 25 or the scanning started at sector 20, then the Scan Command would be completed in a normal manner.

During the Scan Command data is supplied by either the processor or DMA Controller for comparison against the data read from the diskette. In order to avoid having the OR (Over Run) flag set in Status Register 1, it is necessary to have the data available in less than 27 μs (FM Mode) or 13 μs (MFM Mode). If an Overrun occurs the FDC terminates the command.

SEEK

The read/write head within the FDD is moved from cylinder to cylinder under control of the Seek Command. The FDC compares the PCN (Present Cylinder Number) which is the current head position with the NCN (New Cylinder Number), and performs the following operation if there is a difference:

PCN < NCN: Direction signal to FDD set to a 1 (high), and Step Pulses are issued. (Step In.)

PCN > NCN: Direction signal to FDD set to a 0 (low), and Step Pulses are issued. (Step Out.)

The rate at which Step Pulses are issued is controlled by SRT (Stepping Rate Time) in the SPECIFY Command. After each Step Pulse is issued NCN is compared against PCN, and when NCN = PCN, then the SE (Seek End) flag is set in Status Register 0 to a 1 (high), and the command is terminated.

During the Command Phase of the Seek operation the FDC is in the FDC BUSY state, but during the Execution Phase it is in the NON BUSY state. While the FDC is in the NON BUSY state, another Seek Command may be issued, and in this manner parallel seek operations may be done on up to 4 Drives at once.

If an FDD is in a NOT READY state at the beginning of the command execution phase or during the seek operation, then the NR (NOT READY) flag is set in Status Register 0 to a 1 (high), and the command is terminated.

Note that the 8272A Read and Write Commands do not have implied Seeks. Any R/W command should be preceded by: 1) Seek Command; 2) Sense Interrupt Status; and 3) Read ID.

RECALIBRATE

This command causes the read/write head within the FDD to retract to the Track 0 position. The FDC clears the contents of the PCN counter, and checks the status of the Track 0 signal from the FDD. As long as the Track 0 signal is low, the Direction signal remains 1 (high) and Step Pulses are issued. When the Track 0 signal goes high, the SE (SEEK END) flag in Status Register 0 is set to a 1 (high) and the command is terminated. If the Track 0 signal is still low after 77 Step Pulses have been issued, the FDC sets the SE (SEEK END) and EC (EQUIPMENT CHECK) flags of Status Register 0 to both 1s (highs), and terminates the command.

The ability to overlap RECALIBRATE Commands to multiple FDDs, and the loss of the READY signal, as described in the SEEK Command, also applies to the RECALIBRATE Command.

SENSE INTERRUPT STATUS

An Interrupt signal is generated by the FDC for one of the following reasons:

1. Upon entering the Result Phase of:
   a. Read Data Command
   b. Read a Track Command
   c. Read ID Command
   d. Read Deleted Data Command
   e. Write Data Command
   f. Format a Cylinder Command
   g. Write Deleted Data Command
   h. Scan Commands
2. Ready Line of FDD changes state
3. End of Seek or Recalibrate Command
4. During Execution Phase in the NON-DMA Mode

Interrupts caused by reasons 1 and 4 above occur during normal command operations and are easily discernible by the processor. However, interrupts caused by reasons 2 and 3 above may be uniquely identified with the aid of the Sense Interrupt Status Command. This command when issued resets the interrupt signal and via bits 5, 6, and 7 of Status Register 0 identifies the cause of the interrupt.

Neither the Seek or Recalibrate Command have a Result Phase. Therefore, it is mandatory to use the Sense Interrupt Status Command after these commands to effectively terminate them and to provide verification of the head position (PCN).

Table 11. Seek, Interrupt Codes

SEEK END BIT 5	INTERRUPT CODE		CAUSE
	BIT 6	BIT 7	
0	1	1	Ready Line changed state, either polarity
1	0	0	Normal Termination of Seek or Recalibrate Command
1	1	0	Abnormal Termination of Seek or Recalibrate Command

SPECIFY

The Specify Command sets the initial values for each of the three internal timers. The HUT (Head Unload Time) defines the time from the end of the Execution Phase of one of the Read/Write Commands to the head unload state. This timer is programmable from 16 to 240 ms in increments of 16 ms (01 = 16 ms, 02 = 32 ms . . . . OF = 240 ms). The SRT (Step Rate Time) defines the time interval between adjacent step pulses. This timer is programmable from 1 to 16 ms in increments of 1 ms (F = 1 ms, E = 2 ms, D = 3 ms, etc.). The HLT (Head Load Time) defines the time between when the Head Load signal goes high and when the Read/Write operation starts. This timer is programmable from 2 to 254 ms in increments of 2 ms (01 = 2 ms, 02 = 4 ms, 03 = 6 ms . . . . FE = 254 ms).

The step rate should be programmed 1 mS longer than the minimum time required by the drive.

The time intervals mentioned above are a direct function of the clock (CLK on pin 19). Times indicated above are for an 8 MHz clock, if the clock was reduced to 4 MHz (mini-floppy application) then all time intervals are increased by a factor of 2.

The choice of DMA or NON-DMA operation is made by the ND (NON-DMA) bit. When this bit is high (ND = 1) the NON-DMA mode is selected, and when ND = 0 the DMA mode is selected.

SENSE DRIVE STATUS

This command may be used by the processor whenever it wishes to obtain the status of the FDDs. Status Register 3 contains the Drive Status information.

INVALID

If an invalid command is sent to the FDC (a command not defined above), then the FDC will terminate the command. No interrupt is generated by the 8272A during this condition. Bit 6 and bit 7 (DIO and RQM) in the Main Status Register are both high ("1") indicating to the processor that the 8272A is in the Result Phase and the contents of Status Register 0 (ST0) must be read. When the processor reads Status Register 0 it will find an 80H indicating an invalid command was received.

A Sense Interrupt Status Command must be sent after a Seek or Recalibrate interrupt, otherwise the FDC will consider the next command to be an Invalid Command.

In some applications the user may wish to use this command as a No-Op command, to place the FDC in a standby or no operation state.

Table 12. Status Registers

BIT NO.	NAME	SYMBOL	DESCRIPTION
			STATUS REGISTER 0
D_7	Interrupt Code	IC	$D_7=0$ and $D_6=0$ Normal Termination of Command, (NT). Command was completed and properly executed.
D_6			$D_7=0$ and $D_6=1$ Abnormal Termination of Command, (AT). Execution of Command was started, but was not successfully completed.
			$D_7=1$ and $D_6=0$ Invalid Command issue, (IC). Command which was issued was never started.
			$D_7=1$ and $D_6=1$ Abnormal Termination because during command execution the ready signal from FDD changed state.
D_5	Seek End	SE	When the FDC completes the SEEK Command, this flag is set to 1 (high).
D_4	Equipment Check	EC	If a fault Signal is received from the FDD, or if the Track 0 Signal fails to occur after 77 Step Pulses (Recalibrate Command) then this flag is set.
D_3	Not Ready	NR	When the FDD is in the not-ready state and a read or write command is issued, this flag is set. If a read or write command is issued to Side 1 of a single sided drive, then this flag is set.
D_2	Head Address	HD	This flag is used to indicate the state of the head at Interrupt.
D_1	Unit Select 1	US 1	These flags are used to indicate a Drive Unit Number at Interrupt
D_0	Unit Select 0	US 0	
			STATUS REGISTER 1
D_7	End of Cylinder	EN	When the FDC tries to access a Sector beyond the final Sector of a Cylinder, this flag is set.
D_6			Not used. This bit is always 0 (low).
D_5	Data Error	DE	When the FDC detects a CRC error in either the ID field or the data field, this flag is set.
D_4	Over Run	OR	If the FDC is not serviced by the main-systems during data transfers, within a certain time interval, this flag is set.
D_3			Not used. This bit always 0 (low).
D_2	No Data	ND	During execution of READ DATA, WRITE DELETED DATA or SCAN Command, if the FDC cannot find the Sector specified in the IDR Register, this flag is set.
			During executing the READ ID Command, if the FDC cannot read the ID field without an error, then this flag is set.
			During the execution of the READ A Cylinder Command, if the starting sector cannot be found, then this flag is set.
			STATUS REGISTER 1 (CONT.)
D_1	Not Writable	NW	During execution of WRITE DATA, WRITE DELETED DATA or Format A Cylinder Command, if the FDC detects a write protect signal from the FDD, then this flag is set.
D_0	Missing Address Mark	MA	If the FDC cannot detect the ID Address Mark after encountering the index hole twice, then this flag is set.
			If the FDC cannot detect the Data Address Mark or Deleted Data Address Mark, this flag is set. Also at the same time, the MD (Missing Address Mark in Data Field) of Status Register 2 is set.
			STATUS REGISTER 2
D_7			Not used. This bit is always 0 (low).
D_6	Control Mark	CM	During executing the READ DATA or SCAN Command, if the FDC encounters a Sector which contains a Deleted Data Address Mark, this flag is set.
D_5	Data Error in Data Field	DD	If the FDC detects a CRC error in the data field then this flag is set.
D_4	Wrong Cylinder	WC	This bit is related with the ND bit, and when the contents of C on the medium is different from that stored in the IDR, this flag is set.
D_3	Scan Equal Hit	SH	During execution, the SCAN Command, if the condition of "equal" is satisfied, this flag is set.
D_2	Scan Not Satisfied	SN	During executing the SCAN Command, if the FDC cannot find a Sector on the cylinder which meets the condition, then this flag is set.
D_1	Bad Cylinder	BC	This bit is related with the ND bit, and when the content of C on the medium is different from that stored in the IDR and the content of C is FF, then this flag is set.
D_0	Missing Address Mark in Data Field	MD	When data is read from the medium, if the FDC cannot find a Data Address Mark or Deleted Data Address Mark, then this flag is set.
			STATUS REGISTER 3
D_7	Fault	FT	This bit is used to indicate the status of the Fault signal from the FDD.
D_6	Write Protected	WP	This bit is used to indicate the status of the Write Protected signal from the FDD.
D_5	Ready	RDY	This bit is used to indicate the status of the Ready signal from the FDD.
D_4	Track 0	T0	This bit is used to indicate the status of the Track 0 signal from the FDD.
D_3	Two Side	TS	This bit is used to indicate the status of the Two Side signal from the FDD.
D_2	Head Address	HD	This bit is used to indicate the status of Side Select signal to the FDD.
D_1	Unit Select 1	US 1	This bit is used to indicate the status of the Unit Select 1 signal to the FDD.
D_0	Unit Select 0	US 0	This bit is used to indicate the status of the Unit Select 0 signal to the FDD.

ABSOLUTE MAXIMUM RATINGS*

Operating Temperature ..... 0°C to +70°C
Storage Temperature ..... −40°C to +125°C
All Output Voltages ..... −0.5 to +7 Volts
All Input Voltages ..... −0.5 to +7 Volts
Supply Voltage V_{CC} ..... −0.5 to +7 Volts
Power Dissipation ..... 1 Watt

*T_A = 25°C

NOTICE: Stress above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

D.C. CHARACTERISTICS (T_A = 0°C to +70°C, V_{CC} = +5V ±10%)

Symbol	Parameter	Limits Min.	Limits Max.	Unit	Test Conditions
V_{IL}	Input Low Voltage	−0.5	0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC} + 0.5	V	
V_{OL}	Output Low Voltage		0.45	V	I_{OL} = 2.0 mA
V_{OH}	Output High Voltage	2.4	V_{CC}	V	I_{OH} = −400 μA
I_{CC}	V_{CC} Supply Current		120	mA	
I_{IL}	Input Load Current (All Input Pins)		10 −10	μA μA	$V_{IN} = V_{CC}$ V_{IN} = 0V
I_{LOH}	High Level Output Leakage Current		10	μA	$V_{OUT} = V_{CC}$
I_{OFL}	Output Float Leakage Current		±10	μA	0.45V ≤ V_{OUT} ≤ V_{CC}

CAPACITANCE (T_A = 25°C, f_c = 1 MHz, V_{CC} = 0V)

Symbol	Parameter	Limits Min.	Limits Max.	Unit	Test Conditions
$C_{IN(\Phi)}$	Clock Input Capacitance		20	pF	All Pins Except Pin Under Test Tied to AC Ground
C_{IN}	Input Capacitance		10	pF	
$C_{I/O}$	Input/Output Capacitance		20	pF	

A.C. CHARACTERISTICS (T_A = 0°C to +70°C, V_{CC} = +5.0V ±10%)

CLOCK TIMING

Symbol	Parameter	Min.	Max.	Unit	Notes
t_{CY}	Clock Period	120	500	ns	Note 5
t_{CH}	Clock High Period	40		ns	Note 4, 5
t_{RST}	Reset Width	14		t_{CY}	

READ CYCLE

t_{AR}	Select Setup to $\overline{RD}$↓	0		ns	
t_{RA}	Select Hold from $\overline{RD}$↑	0		ns	
t_{RR}	$\overline{RD}$ Pulse Width	250		ns	
t_{RD}	Data Delay from $\overline{RD}$↓		200	ns	
t_{DF}	Output Float Delay	20	100	ns	

A.C. CHARACTERISTICS (Continued) ($T_A = 0°C$ to $+70°C$, $V_{CC} = +5.0V \pm 10\%$)

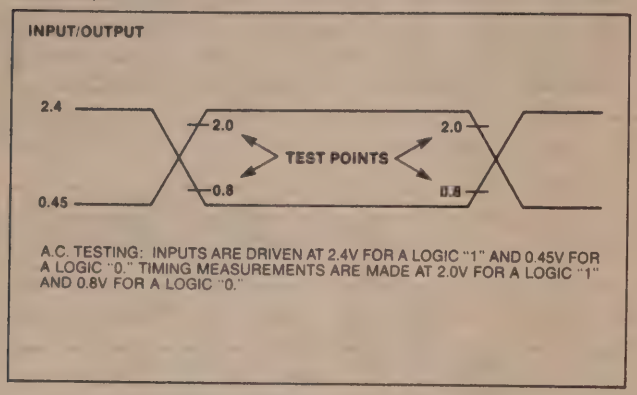
WRITE CYCLE

Symbol	Parameter	Typ.[1]	Min.	Max.	Unit	Notes
tAW	Select Setup to $\overline{WR}$↓		0		ns	
tWA	Select Hold from $\overline{WR}$↑		0		ns	
tWW	$\overline{WR}$ Pulse Width		250		ns	
tDW	Data Setup to $\overline{WR}$↑		150		ns	
tWD	Data Hold from $\overline{WR}$↑		5		ns	
INTERRUPTS						
tRI	INT Delay from $\overline{RD}$↑			500	ns	Note 6
tWI	INT Delay from $\overline{WR}$↑			500	ns	Note 6
DMA						
tRQCY	DRQ Cycle Period		13		μs	Note 6
tAKRQ	$\overline{DACK}$↓ to DRQ↓			200	ns	
tRQR	DRQ↑ to $\overline{RD}$↓		800		ns	Note 6
tRQW	DRQ↑ to $\overline{WR}$↓		250		ns	Note 6
tRQRW	DRQ↑ to $\overline{RD}$↑ or $\overline{WR}$↑			12	μs	Note 6
FDD INTERFACE						
tWCY	WCK Cycle Time	2 or 4 1 or 2			μs	MFM = 0 MFM = 1 Note 2
tWCH	WCK High Time	250	80	350	ns	
tCP	Pre-Shift Delay from WCK↑		20	100	ns	
tCD	WDA Delay from WCK↑		20	100	ns	
tWDD	Write Data Width		tWCH − 50		ns	
tWE	WE↑ to WCK↑ or WE↓ to WCK↓ Delay		20	100	ns	
tWWCY	Window Cycle Time	2 1			μs	MFM = 0 MFM = 1
tWRD	Window Setup to RDD↑		15		ns	
tRDW	Window Hold from RDD↓		15		ns	
tRDD	RDD Active Time (HIGH)		40		ns	
FDD SEEK/DIRECTION/STEP						
tUS	$US_{0,1}$ Setup to $\overline{RW}$/SEEK↑		12		μs	Note 6
tSU	$US_{0,1}$ Hold after $\overline{RW}$/SEEK↓		15		μs	Note 6
tSD	$\overline{RW}$/SEEK Setup to LCT/DIR		7		μs	Note 6
tDS	$\overline{RW}$/SEEK Hold from LCT/DIR		30		μs	Note 6
tDST	LCT/DIR Setup to FR/STEP↑		1		μs	Note 6
tSTD	LCT/DIR Hold from FR/STEP↓		24		μs	Note 6
tSTU	$DS_{2,1}$ Hold from FR/Step↓		5		μs	Note 6
tSTP	STEP Active Time (High)	5			μs	Note 6
tSC	STEP Cycle Time		33		μs	Note 3, 6
tFR	FAULT RESET Active Time (High)		8	10	μs	Note 6
tIDX	INDEX Pulse Width	10			tCY	
tTC	Terminal Count Width		1		tCY	

NOTES:

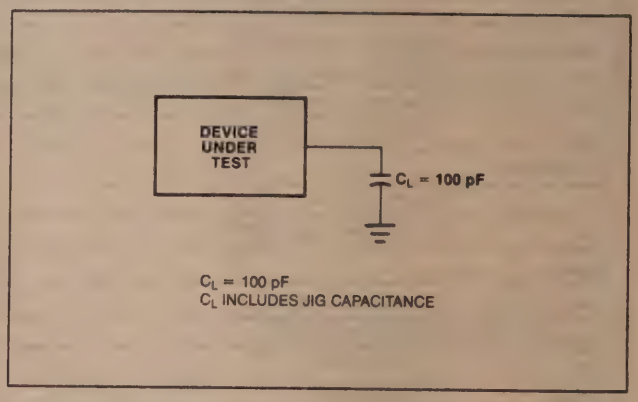
1. Typical values for $T_A = 25°C$ and nominal supply voltage.
2. The former values are used for standard floppy and the latter values are used for mini-floppies.
3. $t_{SC} = 33\ \mu s$ min. is for different drive units. In the case of same unit, t_{SC} can be ranged from 1 ms to 16 ms with 8 MHz clock period, and 2 ms to 32 ms with 4 MHz clock, under software control.
4. From 2.0V to +2.0V.
5. At 4 MHz, the clock duty cycle may range from 16% to 76%. Using an 8 MHz clock the duty cycle can range from 32% to 52%. Duty cycle is defined as: D.C. = 100 (t_{CH} + t_{CY}) with typical rise and fall times of 5 ns.
6. The specified values listed are for an 8 MHz clock period. Multiply timings by 2 when using a 4 MHz clock period.

A.C. TESTING INPUT, OUTPUT WAVEFORM



A.C. TESTING: INPUTS ARE DRIVEN AT 2.4V FOR A LOGIC "1" AND 0.45V FOR A LOGIC "0." TIMING MEASUREMENTS ARE MADE AT 2.0V FOR A LOGIC "1" AND 0.8V FOR A LOGIC "0."

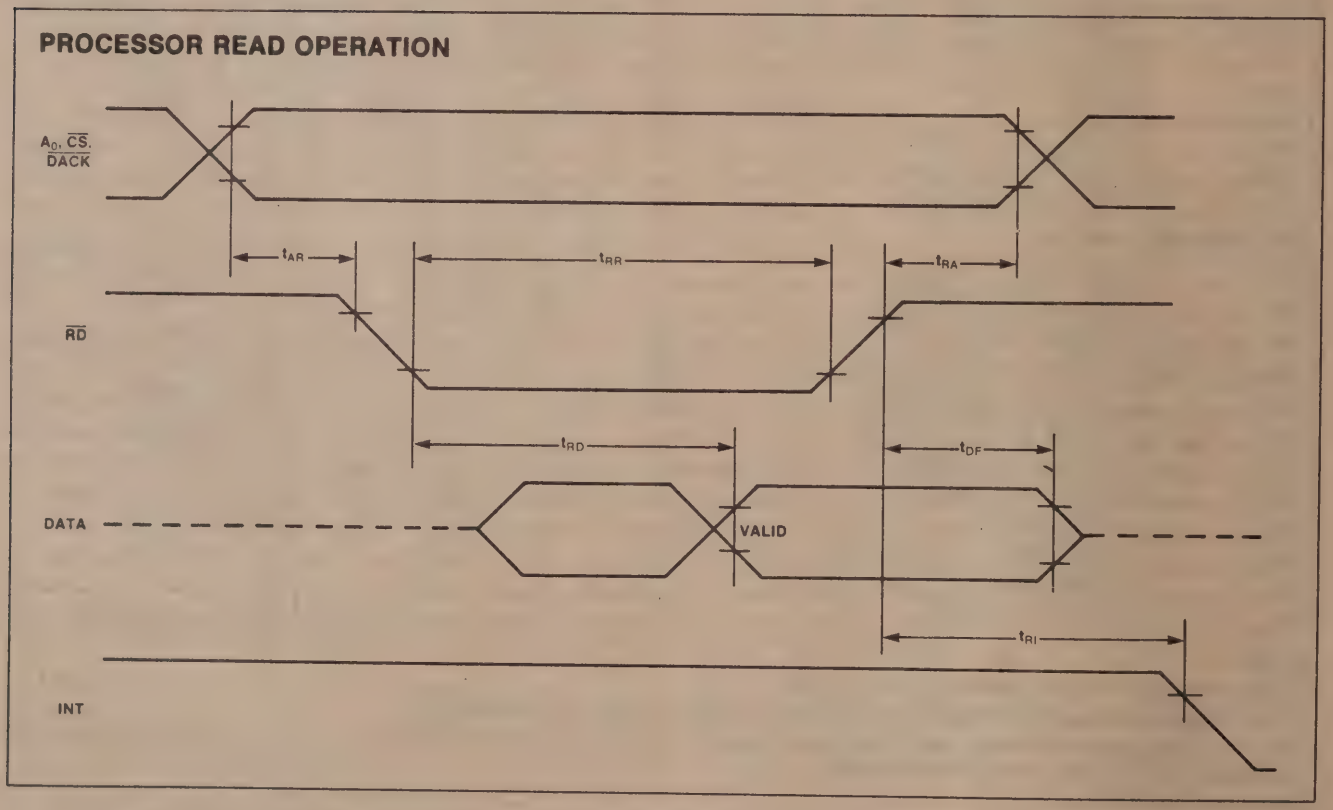
A.C. TESTING LOAD CIRCUIT



C_L = 100 pF
C_L INCLUDES JIG CAPACITANCE

WAVEFORMS

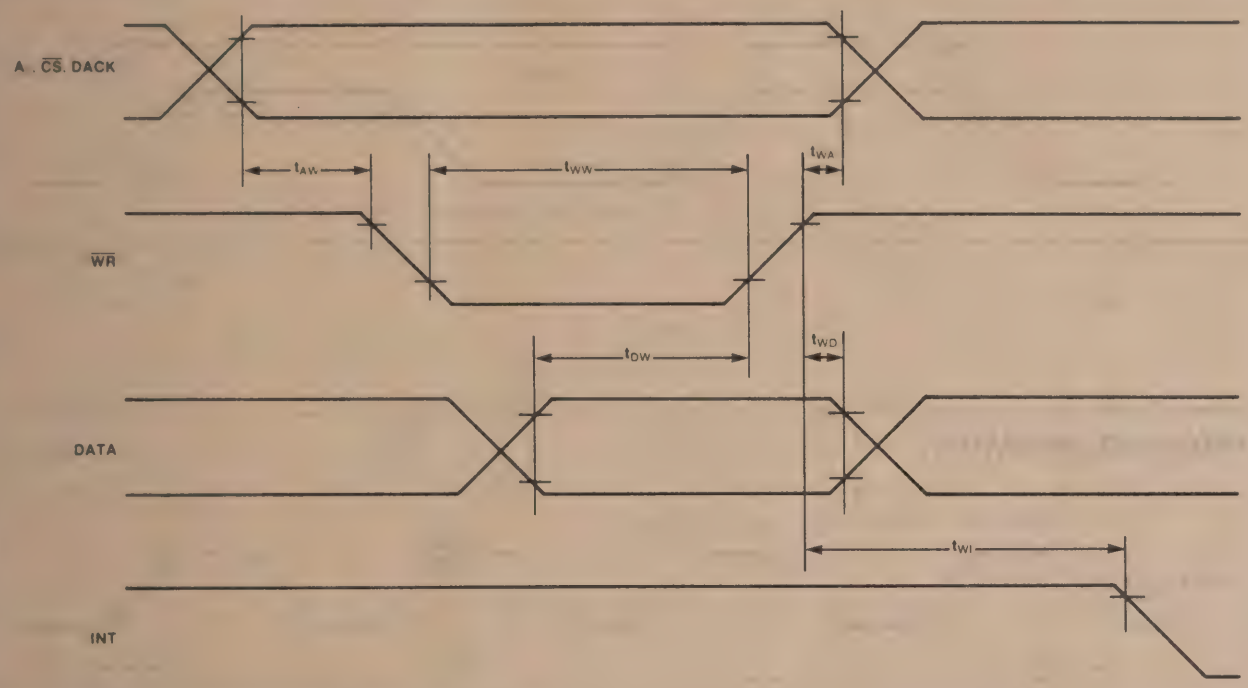
PROCESSOR READ OPERATION



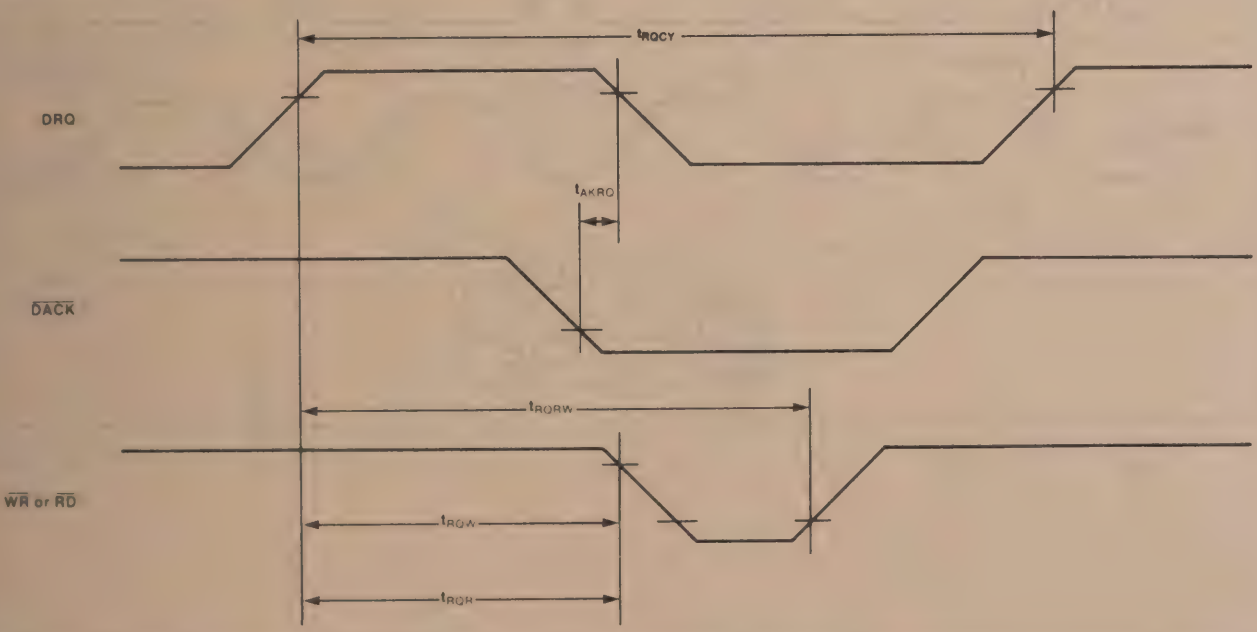
210606-001

WAVEFORMS (Continued)

PROCESSOR WRITE OPERATION

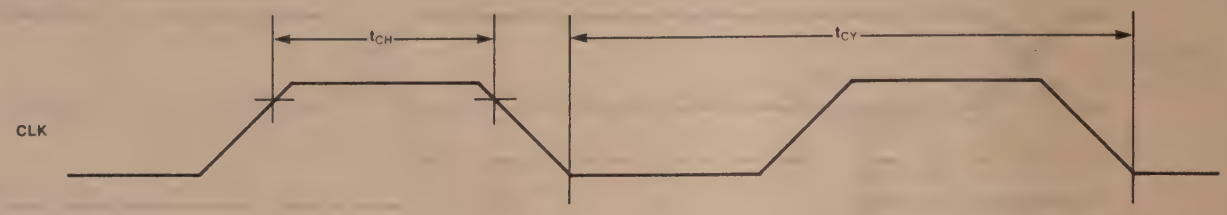


DMA OPERATION

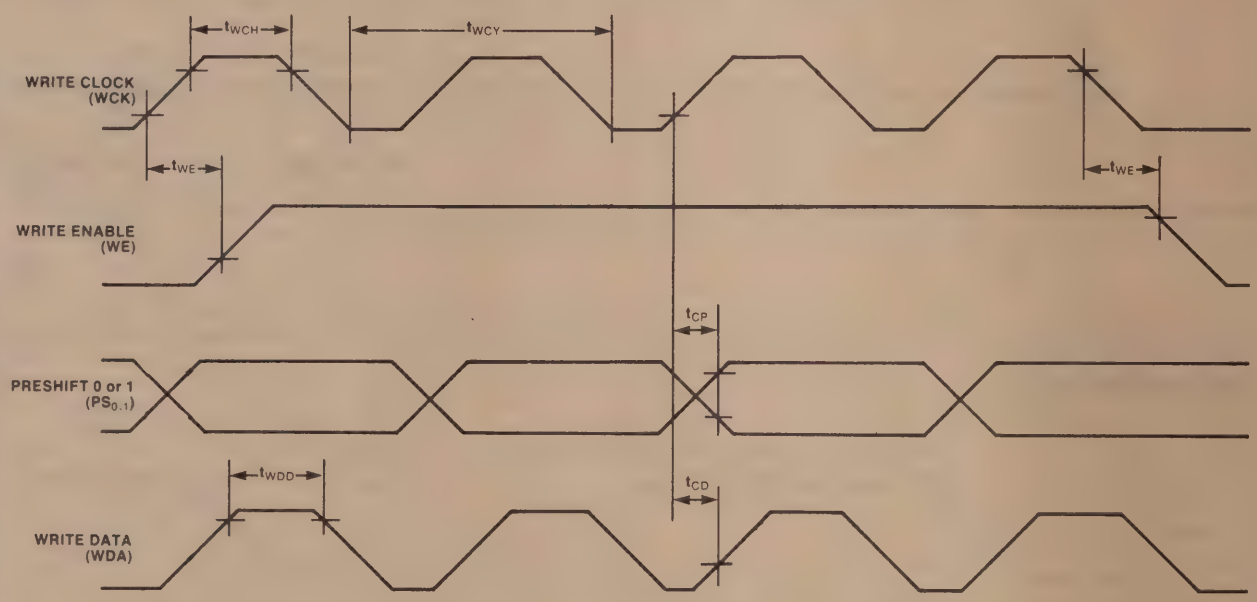


WAVEFORMS (Continued)

CLOCK TIMING



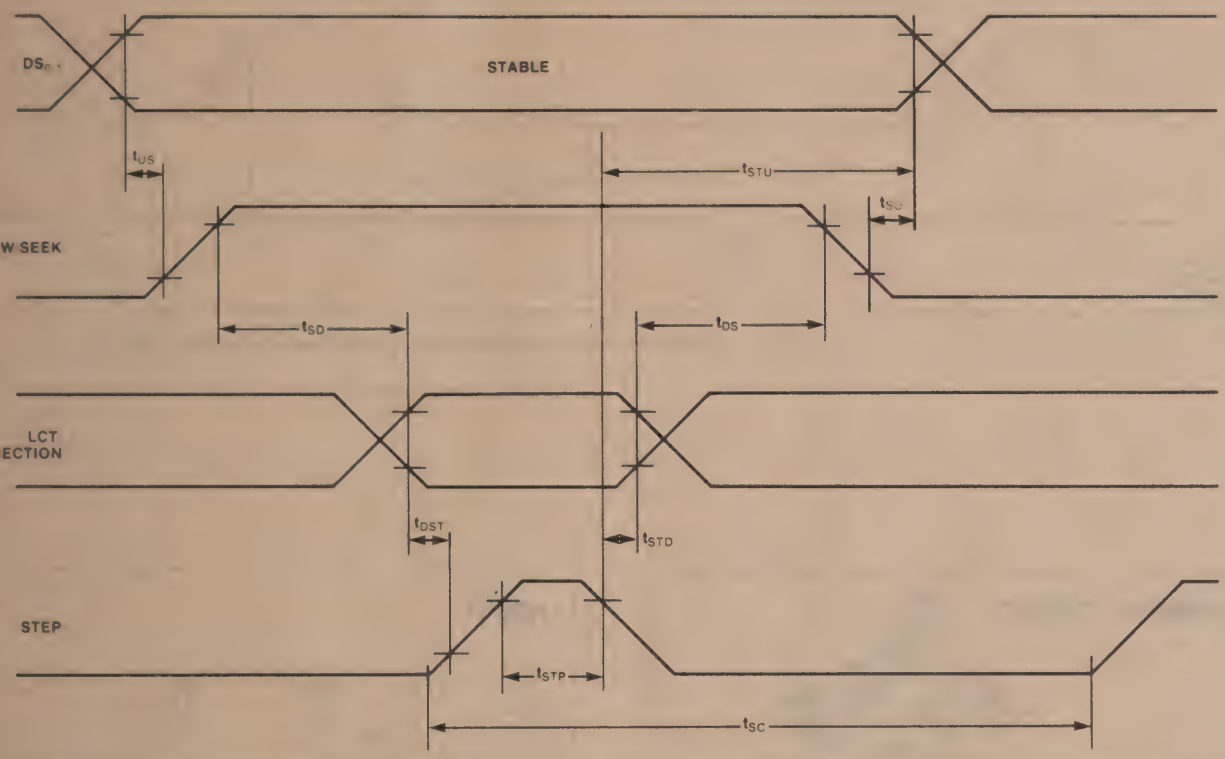
FDD WRITE OPERATION



	PRESHIFT 0	PRESHIFT 1
NORMAL	0	0
LATE	0	1
EARLY	1	0
INVALID	1	1

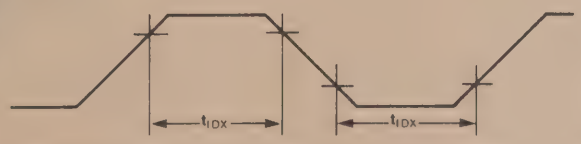
WAVEFORMS (Continued)

SEEK OPERATION



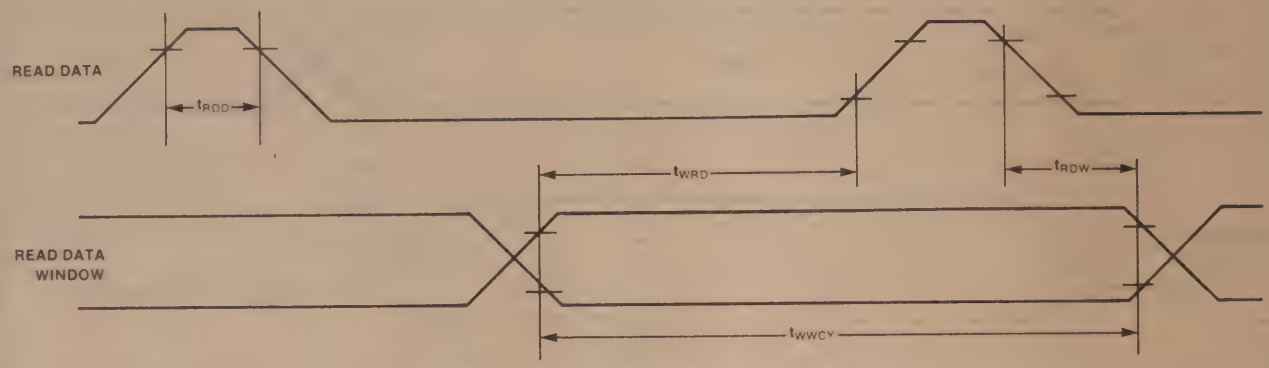
FLT RESET

INDEX

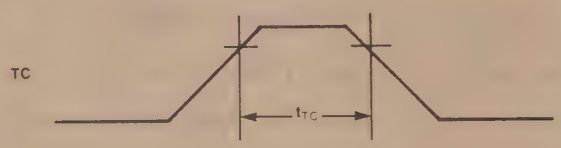


WAVEFORMS (Continued)

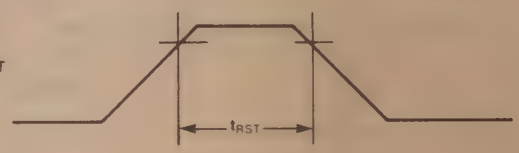
FDD READ OPERATION



TERMINAL COUNT



RESET



intel

APPLICATION NOTE

AP-116

March 1981

An Intelligent Data Base System Using the 8272

Tom Rossi
Peripherals Applications Manager

207875-002

An Intelligent Data Base System Using the 8272

Contents

INTRODUCTION

The Floppy Disk
The Floppy Disk Drive

SUBSYSTEM OVERVIEW

Controller Electronics
Drive Electronics
Controller/Drive Interface
Processor/Memory Interface

DISK FORMAT

Data Recording Techniques
Sectors
Tracks
Sector Interleaving

THE 8272 FLEXIBLE DISKETTE CONTROLLER

Floppy Disk Commands
Interface Registers
Command/Result Phases
Execution Phase
Multi-sector and Multi-track Transfers
Drive Status Polling
Command Details

THE DATA SEPARATOR

Single Density
Double Density
Phase-Locked Loop Design
Initialization
Floppy Disk Data
Startup
PLL Synchronization

AN INTELLIGENT DISKETTE DATA BASE SYSTEM

Processor and Memory
Serial I/O
DMA
Disk Drive Interface

SPECIAL CONSIDERATIONS

APPENDIX

Schematics
Power Distribution

1. INTRODUCTION

Most microcomputer systems in use today require low-cost, high-density removable magnetic media for information storage. In the area of removable media, a designer's choice is limited to magnetic tapes and floppy disks (flexible diskettes), both of which offer non-volatile data storage. The choice between these two technologies is relatively straight-forward for a given application. Since disk drives are designed to permit random access to stored information, they are significantly faster than tape units. For example, locating information on a disk requires less than a second, while tape movement (even at the fastest rewind or fast-forward speed) often requires several minutes. This random access ability permits the use of floppy disks in on-line storage applications (where information must be located, read, and modified/updated in real-time under program or operator control). Tapes, on the other hand, are ideally suited to archival or back-up storage due to their large storage capacities (more than 10 million bytes of data can be archived on a cartridge tape).

A sophisticated controller is required to capitalize on the abilities of the disk storage unit. In the past, disk controller designs have required upwards of 150 ICs. Today, the single-chip 8272 Floppy Disk Controller (FDC) plus approximately 30 support devices can handle up to four million bytes of on-line data storage on four floppy disk drives.

The Floppy Disk

A floppy disk is a circular piece of thin plastic material covered with a magnetic coating and enclosed in a protective jacket (Figure 1). The circular piece of plastic revolves at a fixed speed (approximately 360 rpm) within its jacket in much the same manner that a record revolves at a fixed speed on a stereo turntable. Disks are manufactured in a variety of configurations for various storage capacities. Two standard physical disk sizes are commonly used. The 8-inch disk (8 inches square) is the larger of the two sizes; the smaller size (5-1/4 inches square) is often referred to as a mini-floppy. Single-sided disks can record information on only one side of the disk, while double-sided disks increase the storage capacity by recording on both sides. In addition, disks are classified as single-density or double-density. Double-density disks use a modified recording method to store twice as much information in the same disk area as can be stored on a single-density disk. Table 1 lists storage capacities for standard floppy disk media.

A magnetic head assembly (in contact with the disk) writes information onto the disk surface and subsequently reads the data back. This head assembly can move from the outside edge of the disk toward the center in fixed increments. Once the head assembly is positioned at one of these fixed positions, the head can read or write information in a circular path as the disk revolves beneath the head assembly. This method divides the surface into a fixed number of cylinders (as shown in Figure 2). There are normally 77 cylinders on a standard disk. Once the head assembly is positioned at a given cylinder, data may be read or written on either side of the disk. The appropriate side of the disk is selected by the read/write head address (zero or one). Of course, a single-sided disk can only use head zero. The combination of cylinder address and head address uniquely specifies a single circular track on the disk. The physical beginning of a track is located by means of a small hole (physical index mark) punched through the plastic near the center of the disk. This hole is optically sensed by the drive on every revolution of the disk.

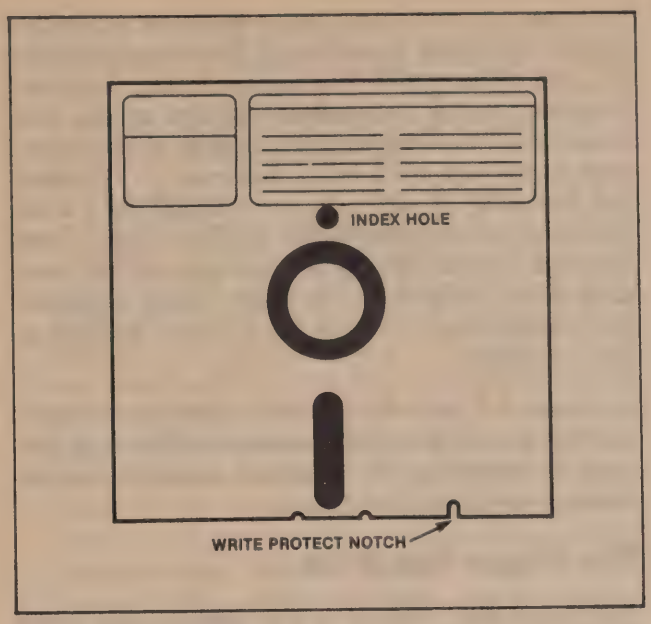


Figure 1. A Floppy Diskette

Table 1. Formatted Disk Capacities

Single-Density Format				
Byte/Sector	128	256	512	1024
Sectors/Track	26	15	8	4
Tracks/Disk	77	77	77	77
Bytes/Disk	256,256	295,680	315,392	315,392
Double-Density Format				
Bytes/Sector	128	256	512	1024
Sectors/Track	52	30	16	8
Tracks/Disk	77	77	77	77
Bytes/Disk	512,512	591,360	630,784	630,784

Each track is subdivided into a number of sectors (see detailed discussion in section 3). Sectors are generally 128, 256, 512, or 1024 data bytes in length. This track sectoring may be accomplished by one of two techniques: hard sectoring or soft sectoring. Hard sectored disks divide each track into a maximum of 32 sectors. The beginning of each sector is indicated by a sector hole punched in the disk plastic. Soft sectoring, the IBM standard method, allows software selection of sector sizes. With this technique, each data sector is preceded by a unique sector identifier that is read/written by the disk controller.

A floppy disk may also contain a write protect notch punched at the edge of the outer jacket of the disk. This notch is detected by the drive and passed to the controller as a write protect signal.

The Floppy Disk Drive

The floppy disk drive is an electromechanical device that records data on, or reads data from, the surface of a floppy disk. The disk drive contains head control electronics that move the head assembly one increment (step) forward (toward the center of the disk) or backward (toward the edge of the disk). Since the recording head must be in contact with the disk material in order to read or write information, the disk drive also contains head-load electronics. Normally the read/write head is unloaded until it is necessary to read or write information on the floppy disk. Once the head assembly has been positioned over the correct track on the disk, the head is loaded (brought into contact with the disk). This sequence prevents excessive disk wear. A small time penalty is paid when the head is loaded. Approximately thirty to fifty milliseconds are needed before data may be reliably read from, or written to, the disk. This time is known as the head load time. If desired, the head may be moved from cylinder to cylinder while loaded. In this manner, only a small time interval (head settling time) is required before data may be read from the new cylinder. The head settling time is often shorter than the head load time. Typically, disk drives also contain drive select logic that allows more than one physical drive to be connected to the same interface cable (from the controller). By means of a jumper on the drive, the drive number may be selected by the OEM or end user. The drive is enabled only when selected; when not selected, all control signals on the cable are ignored.

Finally, the drive provides additional signals to the system controller regarding the status of the drive and disk. These signals include:

Drive Ready — Signals the system that the drive door is closed and that a floppy disk is inserted into the drive.

Track Zero — Indicates that the head assembly is located over the outermost track of the disk. This signal may be used for calibration of the disk drive at system initialization and after an error condition.

Write Protect — Indicates that the floppy disk loaded into the drive is write protected.

Dual Sided — Indicates that the floppy disk in the drive is dual-sided.

Write Fault — Indicates that an error occurred during a recording operation.

Index — Informs the system that the physical index mark of the floppy disk (signifying the start of a data track) has been sensed.

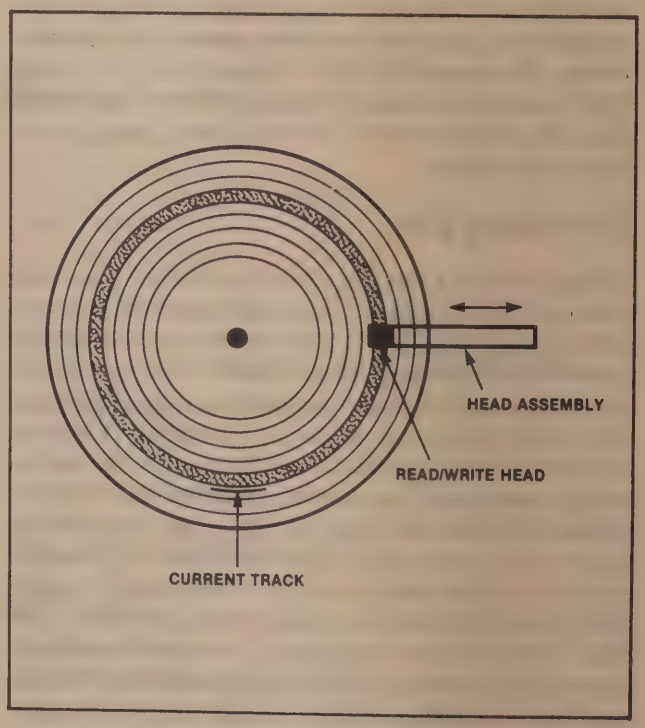


Figure 2. Concentric Cylinders on a Floppy Diskette

2. SUBSYSTEM OVERVIEW

A disk subsystem consists of the following functional electronic units:

1. Disk Controller Electronics
2. Disk Drive Electronics
3. Controller/Disk Interface (cables, drivers, terminators)
4. Controller/Microprocessor System Interface

The operation of these functional units is discussed in the following paragraphs.

Controller Electronics

The disk controller is responsible for converting high-level disk commands (normally issued by software executing on the system processor) into disk drive commands. This function includes:

1. Disk Drive Selection — Disk controllers typically manage the operations of multiple floppy disk drives. This controller function permits the system processor to specify which drive is to be used in a particular operation.
2. Track Selection — The controller issues a timed sequence of step pulses to move the head from its current location to the proper disk cylinder from which data is to be read or to which data is to be written. The controller stores the current cylinder number and computes the stepping distance from the current cylinder to the specified cylinder. The controller also manages the head select signal to select the correct side of the floppy disk.
3. Sector Selection — The controller monitors the data on a track until the requested sector is sensed.
4. Head Loading — The disk controller determines the times at which the head assembly is to be brought in contact with the disk surface in order to read or write data. The controller is also responsible for waiting until the head has settled before reading or writing information. Often the controller maintains the head loaded condition for up to 16 disk revolutions (approximately 2 seconds) after a read or write operation has been completed. This feature eliminates the head load time during periods of heavy disk I/O activity.
5. Data Separation — The actual signal recorded on a floppy disk is a combination of timing information (clock) and data. The serial READ DATA input (from the disk drive) must be converted into two signal streams: clock and data. (The READ DATA input operates at 250K bits/second for single-density disks and 500K bits/second for double-density disks.) The serial data must also be assembled into 8-bit bytes for transfer to system memory. A byte must be assembled and transferred every 32 microseconds for single-density disks and every 16 microseconds for double-density.
6. Error Checking — Information recorded on a floppy disk is subject to both hard and soft errors. Hard (permanent) errors are caused by media defects. Soft errors, on the other hand, are temporary errors caused by electromagnetic noise or mechanical interference. Disk controllers use a standard error checking technique known as a Cyclic Redundancy Check (CRC). As data is written to a disk, a 16-bit CRC character is computed and also stored on the disk. When the data is subsequently read, the CRC character allows the controller to detect data errors. Typically, when CRC errors are detected, the controlling software retries the failed operation (attempting to recover from a soft error). If data cannot reliably be read or written after a number of retries, the system software normally reports the error to the operator. Multiple CRC errors normally indicate unrecoverable media error on the current disk track. Subsequent recovery attempts must be defined by the system designers and tailored to meet system interfacing requirements.

Today, single-chip digital LSI floppy disk controllers such as the 8272 perform all the above functions with the exception of data separation. A data separation circuit (a combination of digital and analog electronics) synchronizes itself to the actual data rate of the disk drive. This data rate varies from drive to drive (due to mechanical factors such as motor tolerances) and varies from disk to disk (due to temperature effects). In order to operate reliably with both single- and double-density storage, the data separation circuit must be based on phase-locked loop (PLL) technology. The phase-locked loop data separation logic is described in section 5. The separation logic, after synchronizing with the data stream, supplies a data window to the LSI disk controller. This window differentiates data information from clock information within the serial stream. The controller uses this window to reconstruct the data previously recorded on the floppy disk.

Drive Electronics

Each floppy disk drive contains digital electronic circuits that translate TTL-compatible command signals into electromechanical operations (such as drive selection and head movement/loading) and that sense and report disk or drive status to the controller (e.g., drive ready, write fault, and write protect). In addition, the drive electronics contain analog components to sense, amplify, and shape data pulses read from, or written to, the floppy disk surface by the read/write head.

Controller/Drive Interface

The controller/drive interface consists of high-current line drivers, Schmitt triggered input gates, and flat or twisted pair cable(s) to connect the disk drive electronics to the controller electronics. Each interface signal line is resistively terminated at the end of the cable farthest from the line drivers. Eight-inch drives may be directly interfaced by means of 50-conductor flat cable. Generally, cable lengths should be less than ten feet in order to maintain noise immunity.

Normally, provisions are made for up to four disk drives to share the same interface cable. The controller may operate as many cable assemblies as practical. LSI floppy disk controllers typically operate one to four drives on a single cable.

Processor/Memory Interface

The disk controller must interface to the system processor and memory for two distinct purposes. First, the processor must specify disk control and command parameters to the controller. These parameters include the selection of the recording density and specification of disk formatting information (discussed in section 3). In addition to disk parameter specification, the processor must also send commands (e.g., read, write, seek, and scan) to the controller. These commands require the specification of the command code, drive number, cylinder address, sector address, and head address. Most LSI controllers receive commands and parameters by means of processor I/O instructions.

In addition to this I/O interface, the controller must also be designed for high-speed data transfer between memory and the disk drive. Two implementation methods may be used to coordinate this data transfer. The lowest-cost method requires direct processor intervention in the transfer. With this method, the controller issues an interrupt to the processor for each data transfer. (An equivalent method allows the processor to poll an interrupt flag in the controller status word.) In the case of a disk write operation, the processor writes a data byte (to be encoded into the serial output stream) to the disk controller following the receipt of each controller interrupt. During a disk read operation, the processor reads a data byte (previously assembled from the input data stream) from the controller after each interrupt. The processor must transfer a data byte from the controller to memory or transfer a data byte from memory to the disk controller within 16 or 32 microseconds after each interrupt (double-density and single-density response times, respectively).

If the system processor must service a variety of other interrupt sources, this interrupt method may not be practical, especially in double-density systems. In this case, the disk controller may be interfaced to a Direct Memory Access (DMA) controller. When the disk controller requires the transfer of a data byte, it simply activates the DMA request line. The DMA controller interfaces to the processor and, in response to the disk controller's request, gains control of the memory interface for a short period of time—long enough to transfer the requested data byte to/from memory. See section 6 for a detailed DMA interface description.

3. DISK FORMAT

New floppy disks must be written with a fixed format by the controller before these disks may be used to store data. Formatting is a method of taking raw media and adding the necessary information to permit the controller to read and write data without error. All formatting is performed by the disk controller on a track-by-track basis under the direction of the system processor. Generally, a track may be formatted at any time. However, since formatting "initializes" a complete disk track, all previously written data is lost (after a format operation). A format operation is normally used only when initializing new floppy disks. Since soft-sectoring in such a predominant formatting technique (due to IBM's influence), the following discussion will limit itself to soft-sectored formats.

Data Recording Techniques

Two standard data recording techniques are used to combine clock and data information for storage on a floppy disk. The single-density technique is referred to as FM encoding. In FM encoding (see Figure 3), a double frequency encoding technique is used that inserts a data bit between two adjacent clock bits. (The presence of a data bit represents a binary "one" while the absence of a data bit represents a binary "zero.") The two adjacent clock bits are referred to as a bit cell, and except for unique field identifiers, all clock bits written on the disk are binary "ones." In FM encoding, each data bit is written at the center of the bit cell and the clock bits are written at the leading edge of the bit cell.

The encoding used for double-density recording is termed MFM encoding (for "Modified FM"). In MFM encoding (Figure 3) the data bits are again written at the center of the bit cell. However, a clock bit is written at the leading edge of the bit cell only if no data bit was written in the previous bit cell and no data bit will be written in the present bit cell.

Sectors

Soft-sectored floppy disks divide each track into a number of data sectors. Typically, sector sizes of 128, 256, 512, or 1024 data bytes are permitted. The sector size is specified when the track is initially formatted by the controller. Table 1 lists the single- and double-

density data storage capacities for each of the four sector sizes. Each sector within a track is composed of the following four fields (illustrated in Figure 4):

1. Sector ID Field — This field, consisting of seven bytes, is written only when the track is formatted. The ID field provides the sector identification that is used by the controller when a sector must be read or written. The first byte of the field is the ID address mark, a unique coding that specifies the beginning of the ID field. The second, third, and fourth bytes are the cylinder, head, and sector addresses, respectively, and the fifth byte is the sector length code. The last two bytes are the 16-bit CRC character for the ID field. During formatting, the controller supplies the address mark. The cylinder, head, and sector addresses and the sector length code are supplied to the controller by the processor software. The CRC character is derived by the controller from the data in the first five bytes.
2. Post ID Field Gap — The post ID field gap (gap 2) is written initially when the track is formatted. During subsequent write operations, the drive's write circuitry is enabled within the gap and the trailing bytes of the gap are rewritten each time the sector is updated (written). During subsequent read operations, the trailing bytes of the gap are used to synchronize the data separator logic with the upcoming data field.
3. Data Field — The length (number of data bytes) of the data field is determined by software when the track is formatted. The first byte of the data field is the data address mark, a unique coding that specifies the beginning of the data field. When a sector is to be deleted, (e.g., a hard error on the disk), a deleted data address mark is written in place of the data address mark. The last two bytes of the data field comprise the CRC character.
4. Post Data Field Gap — The post data field gap (gap 3) is written when the track is formatted and separates the preceding data field from the next physical ID field on the track. Note that a post data field gap is not written following the last physical sector on a track. The gap itself contains a program-selectable number of bytes. Following a sector update (write) operation, the drive's write logic is disabled during the gap. The actual size of gap 3 is determined by the maximum number of data bits that can be recorded on a track, the number of sectors per track and the total sector size (data plus overhead information). The gap size must be adjusted so that it is large enough to contain the discontinuity generated on the floppy disk when the write current is turned on or off (at the start or completion of a disk write operation) and to contain a synchronization field for the upcoming ID field (of the next sector). On the other hand, the gaps must be small enough so that the total number of data bits required on the track (sectors plus gaps) is less than the maximum number of data bits that can be recorded on the track. The gap size must be specified for all read, write, and format operations. The gap size used during disk reads and writes must be smaller than the size used to format the disk to avoid the splice points between contiguous physical sectors. Suggested gap sizes are listed in Table 9.

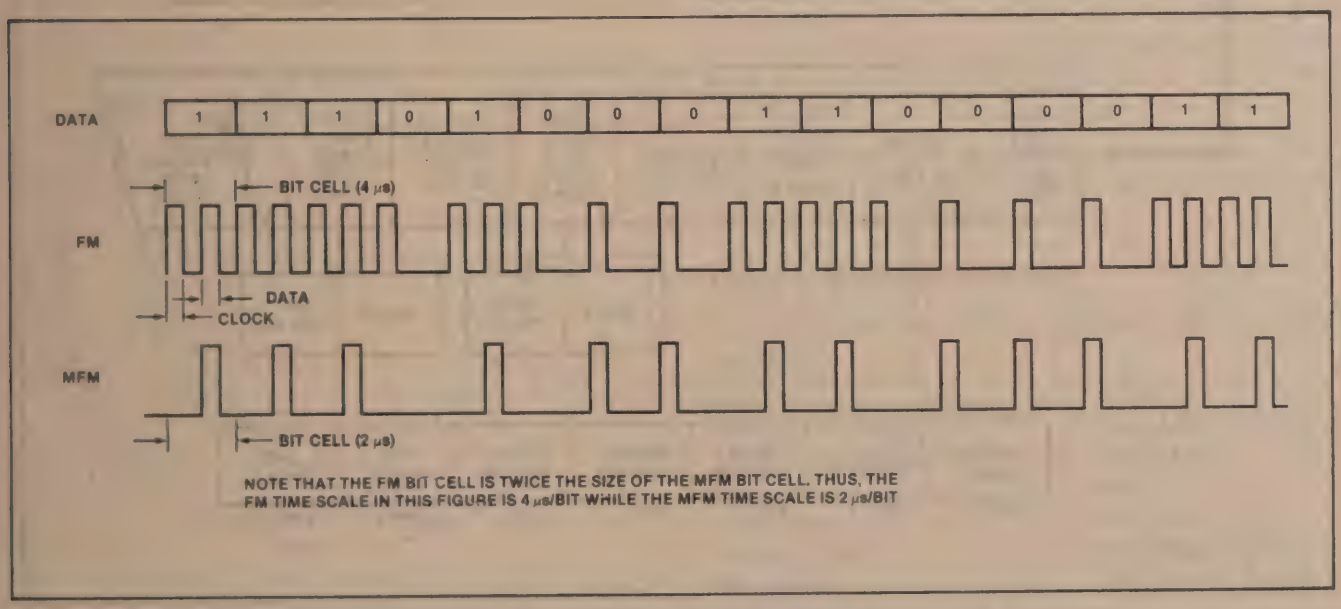


Figure 3. FM and MFM Encoding

207875-002

Tracks

The overall format for a track is illustrated in Figure 4. Each track consists of the following fields:

1. Pre-Index Gap — The pre-index gap (gap 5) is written only when the track is formatted.
2. Index Address Mark — The index address mark consists of a unique code that indicates the beginning of a data track. One index mark is written on each track when the track is formatted.
3. Post Index Gap — The post index gap (gap 1) is used during disk read and write operations to synchronize the data separator logic with the data to be read from the ID field (of the first sector). The post index gap is written only when the disk is formatted.
4. Sectors — The sector information (discussed above) is repeated once for each sector on the track.
5. Final Gap — The final gap (gap 4) is written when the track is formatted and extends from the last physical data field on the track to the physical index mark. The length of this gap is dependent on the number of bytes per sector specified, the lengths of the program-selectable gaps specified, and the drive speed.

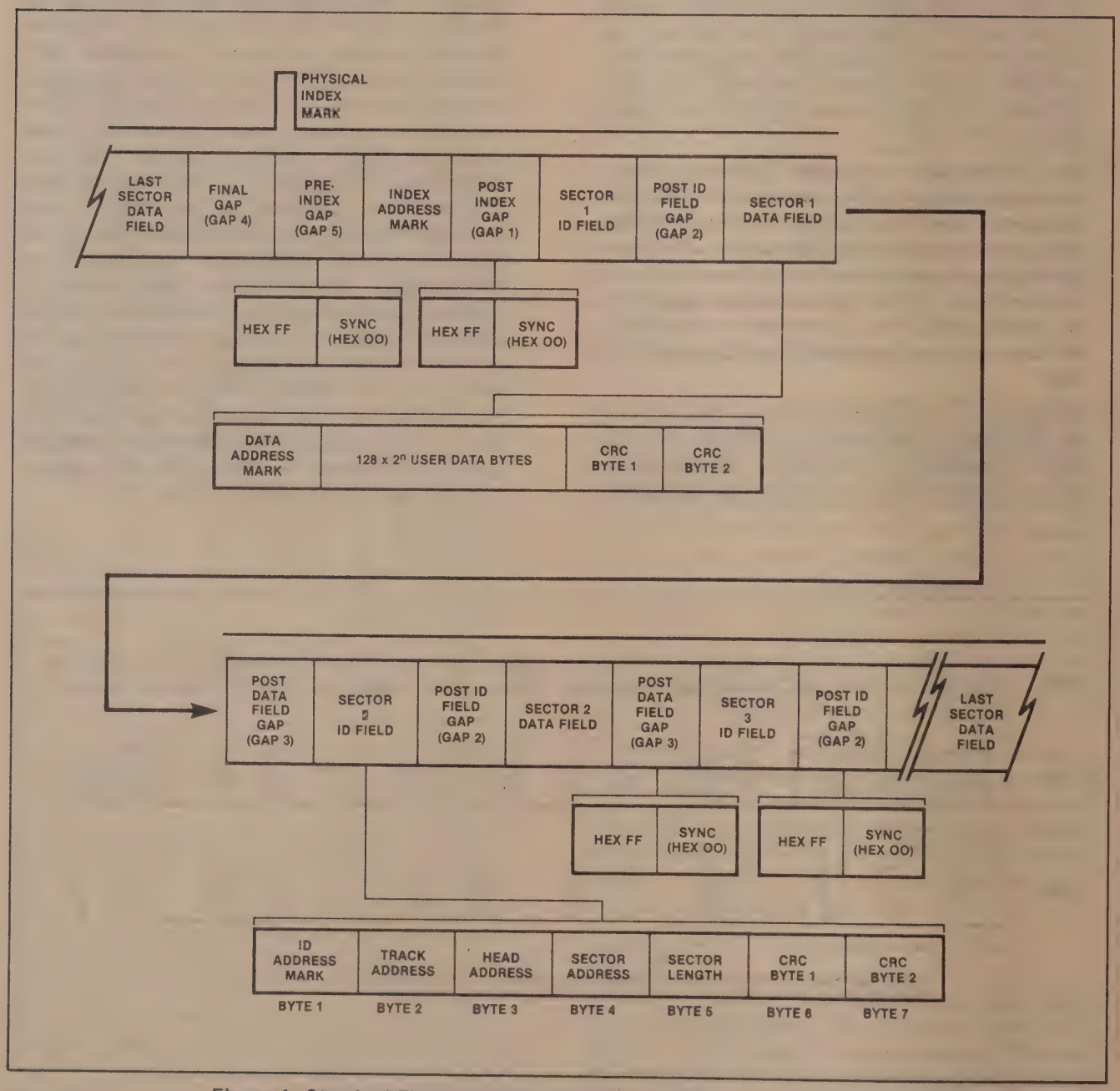


Figure 4. Standard Floppy Diskette Track Format (From SBC 204 Manual)

Sector Interleaving

The initial formatting of a floppy disk determines where sectors are located within a track. It is not necessary to allocate sectors sequentially around the track (i.e., 1,2,3,...,26). In fact, is is often advantageous to place the sectors on the track in a non-sequential order. Sequential sector ordering optimizes sector access times during multi-sector transfers (e.g., when a program is loaded) by permitting the number of sectors specified (up to an entire track) to be transferred within a single revolution of the disk. A technique known as sector interleaving optimizes access times when, although sectors are accessed sequentially, a small amount of processing must be performed between sector reads/writes. For example, an editing program performing a text search reads sectors sequentially, and after each sector is read, performs a software search. If a match is not found, the software issues a read request for the next sector. Since the floppy disk continues to rotate during the time that the software executes, the next physical sector is already passing under the read/write head when the read request is issued, and the processor must wait for another complete revolution of the disk (approximately 166 milliseconds) before the data may actually be input. With interleaving, the sectors are not stored sequentially on a track; rather, each sector is physically removed from the previous sector by some number (known as the interleave factor) of physical sectors as shown in Figure 5. This method of sector allocation provides the processor additional execution time between sectors on the disk. For example, with a 26 sector/track format, an interleave factor of 2 provides 6.4 milliseconds of processing time between sequential 128 byte sector accesses.

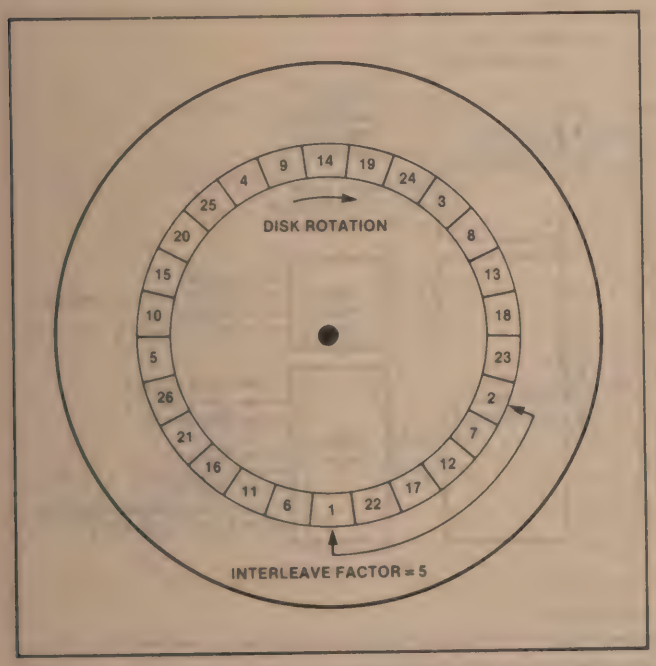


Figure 5. Interleaved Sector Allocation Within a Track

To calculate the correct interleave factor, the maximum processor time between sector operations must be divided by the time required for a complete sector to pass under the disk read/write head. After determining the interleave factor, the correct sector numbers are passed to the disk controller (in the exact order that they are to physically appear on the track) during the execution of a format operation.

4. THE 8272 FLEXIBLE DISKETTE CONTROLLER

The 8272 is a single-chip LSI Floppy Disk Controller (FDC) that contains the circuitry necessary to implement both single-and double-density floppy disk storage subsystems (with up to four dual-sided disk drives per FDC). The 8272 supports the IBM 3740 single-density recording format (FM) and the IBM System 34 double-density recording format (MFM). With the 8272, less than 30 ICs are needed to implement a complete disk subsystem. The 8272 accepts and executes high-level disk commands such as format track, seek, read sector, write sector, and read track. All data synchronization and error checking is automatically performed by the FDC to ensure reliable data storage and subsequent retrieval. External logic is required only for the generation of the FDC master clock and write clock (see Section 6) and for data separation (Section 5). The FDC provides signals that control the startup and base frequency selection of the data separator. These signals greatly ease the design of a phase-locked loop data separator.

In addition to the data separator interface signals, the 8272 also provides the necessary signals to interface to microprocessor systems with or without Direct Memory Access (DMA) capabilities. In order to interface to a large number of commercially available floppy disk drives, the FDC permits software specification of the track stepping rate, the head load time, and the head unload time.

The pin configuration and internal block diagram of the 8272 is shown in Figure 6. Table 2 contains a description for each FDC interface pin.

Floppy Disk Commands

The 8272 executes fifteen high-level disk interface commands:

Specify	Write Data
Sense Drive Status	Write Deleted Data
Sense Interrupt Status	Read Track
Seek	Read ID
Recalibrate	Scan Equal
Format Track	Scan High or Equal
Read Data	Scan Low or Equal
Read Deleted Data	

Each command is initiated by a multi-byte transfer from the processor to the FDC (the transferred bytes contain command and parameter information). After complete command specification, the FDC automatically executes the command. The command result data (after execution of the command) may require a multi-byte transfer of status information back to the processor. It is convenient to consider each FDC command as consisting of the following three phases:

COMMAND PHASE: The executing program transfers to the FDC all the information required to perform a particular disk operation. The 8272 automatically enters the command phase after RESET and following the completion of the result phase (if any) of a previous command.

EXECUTION PHASE: The FDC performs the operation as instructed. The execution phase is entered immediately after the last command parameter is written to the FDC in the preceding command phase. The execution phase normally ends when the last data byte is transferred to/from the disk (signalled by the TC input to the FDC) or when an error occurs.

RESULT PHASE: After completion of the disk operation, status and other housekeeping information are made available to the processor. After the processor reads this information, the FDC reenters the command phase and is ready to accept another command.

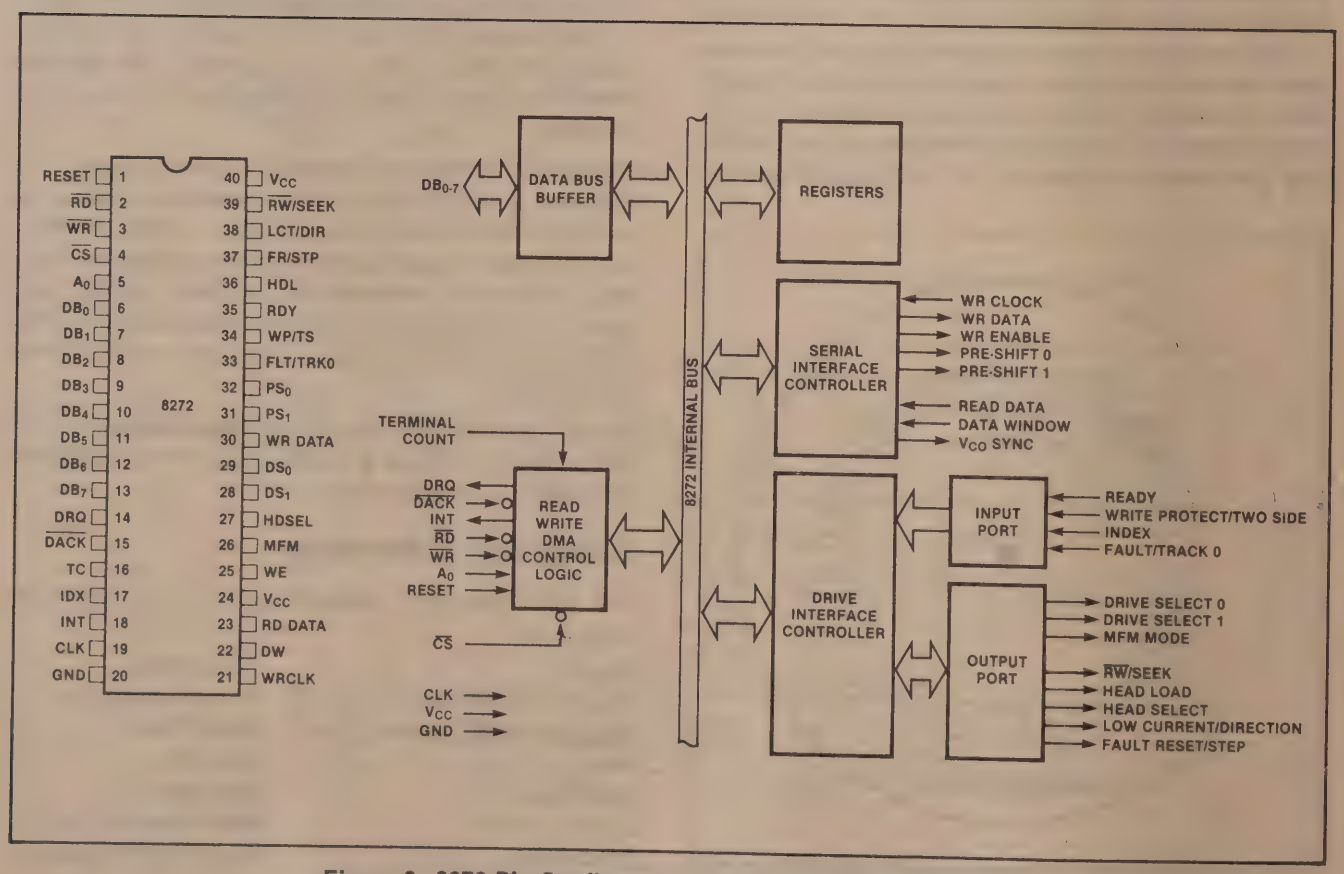


Figure 6. 8272 Pin Configuration and Internal Block Diagram

Table 2. 8272 FDC Pin Description

Number	Pin Symbol	I/O	To/From	Description
1	RST	I	uP	Reset. Active-high signal that places the FDC in the "idle" state and all disk drive output signals are forced inactive (low). This input must be held active during power on reset while the RD and WR inputs are active.
2	$\overline{RD}$	I*	uP	Read. Active-low control signal that enables data transfer from the FDC to the data bus.
3	$\overline{WR}$	I*	uP	Write. Active-low control signal that enables data transfer from the data bus into the FDC.
4	$\overline{CS}$	I	uP	Chip Select. Active-low control signal that selects the FDC. No reading or writing will occur unless the FDC is selected.
5	A_0	I*	uP	Address. Selects the Data Register or Main Status Register for input/output in conjunction with the RD and WR inputs. (See Table 3.)
6-13	DB_0-DB_7	I/O*	uP	Data Bus. Bidirectional three-state 8-bit data bus.
14	DRQ	O	DMA	DMA Request. Active-high output that indicates an FDC request for DMA services.
15	$\overline{DACK}$	I	DMA	DMA Acknowledge. Active-low control signal indicating that the requested DMA transfer is in progress.
16	TC	I	DMA	Terminal Count. Active-high signal that causes the termination of a command. Normally, the terminal count input is directly connected to the TC/EOP output from the DMA controller, signalling that the DMA transfer has been completed. In a non-DMA environment, the processor must count data transfers and supply a TC signal to the FDC.
17	IDX	I	Drive	Index. Indicates detection of the physical index mark (the beginning of a track) on the selected disk drive.
18	INT	O	uP	Interrupt Request. Active-high signal indicating an 8272 interrupt service request.
19	CLK	I		Clock. Signal phase 8 MHz clock (50% duty cycle).
20	GND			Ground. DC power return.
21	WR CLK	I		Write Clock. 500 kHz (FM) or 1 MHz (MFM) write clock with a constant pulse width of 250 ns (for both FM and MFM recording). The write clock must be present at all times.
22	DW	I	PLL	Data Window. Data sample signal from the phase-locked loop indicating that the FDC should sample input data from the disk drive.
23	RD DATA	I	Drive	Read Data. FDC input data from the selected disk drive.
24	VCO	O	PLL	VCO Sync. Active-high output that enables the phase-locked loop to synchronize with the input data from the disk drive.
25	WE	O	Drive	Write Enable. Active-high output that enables the disk drive write gate.
26	MFM	O	PLL	MFM Mode. Active-high output used by external logic to enable the MFM double-density recording mode. When the MFM output is low, single-density FM recording is indicated.
27	HDSEL	O	Drive	Head Select. Selects head 0 or head 1 on a dual-sided disk.
28,29	DS_1,DS_0	O	Drive	Drive Select. Selects one of four disk drives.
30	WR DATA	O	Drive	Write Data. Serial data stream (combination of clock and data bits) to be written on the disk.
31,32	PS_1,PS_0	O	Drive	Precompensation (pre-shift) Control. Write precompensation output control during MFM mode. Specifies early, late, and normal timing signals. See the discussion in Section 5.

Table 2. 8272 FDC Pin Description (continued)

Number	Pin Symbol	I/O	To/From	Description
33	FLT/TRKO	I	Drive	Fault/Track 0. Senses the disk drive fault condition in the Read/Write mode and the Track 0 condition in the Seek mode.
34	WP/TS	I	Drive	Write Protect/Two-Sided. Senses the disk write protect status in the Read/Write mode and the dual-sided media status in the Seek mode.
35	RDY	I	Drive	Ready. Senses the disk drive ready status.
36	HDL	O	Drive	Head Load. Loads the disk drive read/write head. (The head is placed in contact with the disk.)
37	FR/STP	O	Drive	Fault Reset/Step. Resets the fault flip-flop in the disk drive when operating in the Read/Write mode. Provides head step pulses (to move the head from one cylinder to another cylinder) in the Seek mode.
38	LCT/DIR	O	Drive	Low Current/Direction. Signals that the recording head has been positioned over the inner cylinders (44-77) of the floppy disk in the Read/Write mode. (The write current must be lowered when recording on the physically shorter inner cylinders of the disk. Most drives do not track the actual head position and require that the FDC supply this signal.) Determines the head step direction in the Seek mode. In the Seek mode, a high level on this pin steps the read/write head toward the spindle (step-in); a low level steps the head away from the spindle (step-out).
39	RW/SEEK	O	Drive	Read, Write/Seek Mode Selector. A high level selects the Seek mode; a low level selects the Read/Write mode.
40	V_{CC}			+ 5V DC Power.

*Disabled when CS is high.

Interface Registers

To support information transfer between the FDC and the system processor, the 8272 contains two 8-bit registers: the Main Status Register and the Data Register. The Main Status Register (read only) contains FDC status information and may be accessed at any time. The Main Status Register (Table 4) provides the system processor with the status of each disk drive, the status of the FDC, and the status of the processor interface. The Data Register (read/write) stores data, commands, parameters, and disk drive status information. The Data Register is used to program the FDC during the command phase and to obtain result information after completion of FDC operations. Data is read from, or written to, the FDC registers by the combination of the A0, $\overline{RD}$, $\overline{WR}$, and $\overline{CS}$ signals, as described in Table 3.

In addition to the Main Status Register, the FDC contains four additional status registers (ST0, ST1, ST2, and ST3). These registers are only available during the result phase of a command.

Table 3. FDC Read/Write Interface

$\overline{CS}$	A_0	$\overline{RD}$	$\overline{WR}$	Function
0	0	0	1	Read Main Status Register
0	0	1	0	Illegal
0	0	0	0	Illegal
0	1	0	0	Illegal
0	1	0	1	Read from Data Register
0	1	1	0	Write into Data Register
1	X	X	X	Data Bus is three-stated

Table 4. Main Status Register Bit Definitions

Bit Number	Symbol	Description
0	D_0B	Disk Drive 0 Busy. Disk Drive 0 is in the Seek mode.
1	D_1B	Disk Drive 1 Busy. Disk Drive 1 is in the Seek mode.
2	D_2B	Disk Drive 2 Busy. Disk Drive 2 is in the Seek mode.
3	D_3B	Disk Drive 3 Busy. Disk Drive 3 is in the Seek mode.
4	CB	FDC Busy. A read or write command is in process.
5	NDM	Non-DMA Mode. The FDC is in the non-DMA mode when this bit is high. This bit is set only during the execution phase of commands in the non-DMA mode. Transition to a low level indicates that the execution phase has ended.
6	DIO	Data Input/Output. Indicates the direction of a data transfer between the FDC and the Data Register. When DIO is high, data is read from the Data Register by the processor; when DIO is low, data is written from the processor to the Data Register.
7	RQM	Request for Master. Indicates that the Data Register is ready to send data to, or receive data from, the processor.

Command/Result Phases

Table 5 lists the 8272 command set. For each of the fifteen commands, command and result phase data transfers are listed. A list of abbreviations used in the table is given in Table 6, and the contents of the result status registers (ST0-ST3) are illustrated in Table 7.

The bytes of data which are sent to the 8272 during the command phase, and are read out of the 8272 in the result phase, must occur in the order shown in Table 5. That is, the command code must be sent first and the other bytes sent in the prescribed sequence. All bytes of the command and result phases must be read/written as described. After the last byte of data in the command phase is sent to the 8272 the execution phase automatically starts. In a similar fashion, when the last byte of data is read from the 8272 in the result phase, the command is automatically ended and the 8272 is ready for a new command. A command may be aborted by simply raising the terminal count signal (pin 16). This is a convenient means of ensuring that the processor may always gain control of the 8272 (even if the disk system hangs up in an abnormal manner).

It is important to note that during the result phase all bytes shown in Table 5 must be read. The Read Data command, for example, has seven bytes of data in the result phase. All seven bytes must be read in order to successfully complete the Read Data command. The 8272 will not accept a new command until all seven bytes have been read. The number of command and result bytes varies from command-to-command.

In order to read data from, or write data to, the Data Register during the command and result phases, the system processor must examine the Main Status Register to determine if the Data Register is available. The DI0 (bit 6) and RQM (bit 7) flags in the Main Status Register must be low and high, respectively, before each byte of the command word may be written into the 8272. Many of the commands require multiple bytes, and as a result, the Main Status Register must be read prior to each byte transfer to the 8272. To read status bytes during the result phase, DI0 and RQM in the Main Status Register must both be high. Note, checking the Main Status Register in this manner before each byte transfer to/from the 8272 is required only in the command and result phases, and is NOT required during the execution phase.

Execution Phase

All data transfers to (or from) the floppy drive occur during the execution phase. The 8272 has two primary modes of operation for data transfers (selected by the specify command):

1. DMA mode
2. non-DMA mode

In the DMA mode, DRQ (DMA Request) is activated for each transfer request. The DMA controller responds to DRQ with $\overline{DACK}$ (DMA Acknowledge) and $\overline{RD}$ (for read commands) or $\overline{WR}$ (for write commands). DRQ is reset by the FDC during the transfer. INT is activated after the last data transfer, indicating the completion of the execution phase, and the beginning of the result phase. In the DMA mode, the terminal count (TC/EOP) output of the DMA controller should be connected to the 8272 TC input to properly terminate disk data transfer commands.

Table 5. 8272 Command Set

PHASE	R/W	DATA BUS D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	REMARKS
READ DATA										
Command	W	MT	MFM	SK	0	0	1	1	0	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W	C								Sector ID information prior to Command execution
	W	H								
	W	R								
	W	N								
	W	EOT								
	W	GPL								
	W	DTL								
Execution										Data transfer between the FDD and the main-system
Result	R	ST 0								Status information after Command execution
	R	ST 1								
	R	ST 2								
	R	C								Sector ID information after command execution
	R	H								
	R	R								
	R	N								
READ DELETED DATA										
Command	W	MT	MFM	SK	0	1	1	0	0	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W	C								Sector ID information prior to Command execution
	W	H								
	W	R								
	W	N								
	W	EC 1								
	W	GPL								
	W	DTL								
Execution										Data transfer between the FDD and the main-system
Result	R	ST 0								Status information after Command execution
	R	ST 1								
	R	ST 2								
	R	C								Sector ID information after Command execution
	R	H								
	R	R								
	R	N								
WRITE DATA										
Command	W	MT	MFM	0	0	0	1	0	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W	C								Sector ID information prior to Command execution
	W	H								
	W	R								
	W	N								
	W	EOT								
	W	GPL								
	W	DTL								
Execution										Data transfer between the main-system and the FDD
Result	R	ST 0								Status information after Command execution
	R	ST 1								
	R	ST 2								
	R	C								Sector ID information after Command execution
	R	H								
	R	R								
	R	N								
WRITE DELETED DATA										
Command	W	MT	MFM	0	0	1	0	0	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W	C								Sector ID information prior to Command execution
	W	H								
	W	R								
	W	N								
	W	EOT								
	W	GPL								
	W	DTL								
Execution										Data transfer between the FDD and the main-system
Result	R	ST 0								Status information after Command execution
	R	ST 1								
	R	ST 2								
	R	C								Sector ID information after Command execution
	R	H								
	R	R								
	R	N								

PHASE	R/W	DATA BUS D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	REMARKS
READ A TRACK										
Command	W	0	MFM	SK	0	0	0	1	0	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W	C								Sector ID information prior to Command execution
	W	H								
	W	R								
	W	N								
	W	EOT								
	W	GPL								
	W	DTL								
Execution										Data transfer between the FDD and the main-system. FDC reads the complete track contents from the physical index mark to EOT
Result	R	ST 0								Status information after Command execution
	R	ST 1								
	R	ST 2								
	R	C								Sector ID information after Command execution
	R	H								
	R	R								
	R	N								
READ ID										
Command	W	0	MFM	0	0	1	0	1	0	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
Execution										The first correct ID information on the track is stored in Data Register
Result	R	ST 0								Status information after Command execution
	R	ST 1								
	R	ST 2								
	R	C								Sector ID information during Execution Phase
	R	H								
	R	R								
	R	N								
FORMAT A TRACK										
Command	W	0	MFM	0	0	1	1	0	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W	N								Bytes/Sector
	W	SC								Sectors/Track
	W	GPL								Gap 3
	W	D								Filter Byte
Execution										FDC formats an entire track
Result	R	ST 0								Status information after Command execution
	R	ST 1								
	R	ST 2								
	R	C								In this case, the ID information has no meaning
	R	H								
	R	R								
	R	N								
SCAN EQUAL										
Command	W	MT	MFM	SK	1	0	0	0	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W	C								Sector ID information prior to Command execution
	W	H								
	W	R								
	W	N								
	W	EOT								
	W	GPL								
	W	STP								
Execution										Data compared between the FDD and the main-system
Result	R	ST 0								Status information after Command execution
	R	ST 1								
	R	ST 2								
	R	C								Sector ID information after Command execution
	R	H								
	R	R								
	R	N								

Note: 1. $A_0 = 1$ for all operations.

Table 5. Command Set (Continued)

PHASE	R/W	DATA BUS D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	REMARKS
SCAN LOW OR EQUAL										
Command	W	MT	MFM	SK	1	1	0	0	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W	C								Sector ID information prior Command execution
	W	H								
	W	R								
	W	N								
	W	EOT								
	W	GPL								
	W	STP								
Execution										Data compared between the FDD and the main-system
Result	R	ST 0								Status information after Command execution
	R	ST 1								
	R	ST 2								
	R	C								
	R	H								Sector ID information after Command execution
	R	R								
	R	N								
SCAN HIGH OR EQUAL										
Command	W	MT	MFM	SK	1	1	1	0	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W	C								Sector ID information prior Command execution
	W	H								
	W	R								
	W	N								
	W	EOT								
	W	GPL								
	W	STP								
Execution										Data compared between the FDD and the main-system
Result	R	ST 0								Status information after Command execution
	R	ST 1								
	R	ST 2								
	R	C								
	R	H								Sector ID information after Command execution
	R	R								
	R	N								

PHASE	R/W	DATA BUS D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	REMARKS
RECALIBRATE										
Command	W	0	0	0	0	0	1	1	1	Command Codes
	W	0	0	0	0	0	0	DS1	DS0	
Execution										Head retracted to Track 0
SENSE INTERRUPT STATUS										
Command	W	0	0	0	0	1	0	0	0	Command Codes
Result	R	ST 0								Status information at the end of each seek operation about the FDC
	R	C								
SPECIFY										
Command	W	0	0	0	0	0	0	1	1	Command Codes
	W	SPT						HUT		Timer Settings
	W	HLT							ND	
SENSE DRIVE STATUS										
Command	W	0	0	0	0	0	1	0	0	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
Result	R	ST 3								Status information about the FDD
SEEK										
Command	W	0	0	0	0	1	1	1	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W	C								
Execution										Head is positioned over proper Cylinder on Diskette
INVALID										
Command	W	Invalid Codes								Invalid Command Codes (NoOp — FDC goes into Standby State)
Result	R	ST 0								ST 0 = 80 (16)

Table 6. Command/Result Parameter Abbreviations

Symbol	Description
C	Cylinder Address. The currently selected cylinder address (0 to 76) on the disk.
D	Data Pattern. The pattern to be written in each sector data field during formatting.
DS0,DS1	Disk Drive Select. DS1 DS0 0 0 Drive 0 0 1 Drive 1 1 0 Drive 2 1 1 Drive 3
DTL	Special Sector Size. During the execution of disk read/write commands, this parameter is used to temporarily alter the effective disk sector size. By setting N to zero, DTL may be used to specify a sector size from 1 to 256 bytes in length. If the actual sector (on the diskette) is larger than DTL specifies, the remainder of the actual sector is not passed to the system during read commands; during write commands, the remainder of the actual sector is written with all-zeroes bytes. DTL should be set to FF hexadecimal when N is not zero.
EOT	End of Track. The final sector number of the current track.
GPL	Gap Length. The gap 3 size. (Gap 3 is the space between sectors excluding the VCO synchronization field as defined in section 3.)
H	Head Address. Selected head: 0 or 1 (disk side 0 or 1, respectively) as encoded in the sector ID field.
HLT	Head Load Time. Defines the time interval that the FDC waits after loading the head before initiating a read or write operation. Programmable from 2 to 254 milliseconds (in increments of 2 ms).
HUT	Head Unload Time. Defines the time interval from the end of the execution phase (of a read or write command) until the head is unloaded. Programmable from 16 to 240 milliseconds (in increments of 16 ms).
MFM	MFM/FM Mode Selector. Selects MFM double-density recording mode when high, FM single-density mode when low.

Table 6. Command/Result Parameter Abbreviations (continued)

Symbol	Description	Symbol	Description
MT	Multi-Track Selector. When set, this flag selects the multi-track operating mode. In this mode (used only with dual-sided disks), the FDC treats a complete cylinder (under both read/write head 0 and read/write head 1) as a single track. The FDC operates as if this expanded track started at the first sector under head 0 and ended at the last sector under head 1. With this flag set (high), a multi-sector read opeation will automatically continue to the first sector under head 1 when the FDC finishes operating on the last sector under head 0.	SK	Skip Flag. When this flag is set, sectors containing deleted data address marks will automatically be skipped during the execution of multi-sector Read Data or Scan commands. In the same manner, a sector containing a data address mark will automatically be skipped during the execution of a multi-sector Read Deleted Data command.
		SRT	Step Rate Interval. Defines the time interval between step pulses issued by the FDC (track-to-track access time). Programmable from 1 to 16 milliseconds (in increments of 1 ms).
N	Sector Size. The number of data bytes within a sector. (See Table 9.)	ST0 ST1 ST2 ST3	Status Register 0-3. Registers within the FDC that store status information after a command has been executed. This status information is available to the processor during the Result Phase after command execution. These registers may only be read after a command has been executed (in the exact order shown in Table 5 for each command). These registers should not be confused with the Main Status Register.
ND	Non-DMA Mode Flag. When set (high), this flag indicates that the FDC is to operate in the non-DMA mode. In this mode, the processor is interrupted for each data transfer. When low, the FDC interfaces to a DMA controller by means of the DRQ and DACK signals.		
R	Sector Address. Specifies the sector number to be read or written. In multi-sector transfers, this parameter specifies the sector number of the first sector to be read or written.	STP	Scan Sector Increment. During Scan operations, this parameter is added to the current sector number in order to determine the next sector to be scanned.
SC	Number of Sectors per Track. Specifies the number of sectors per track to be initialized by the Format Track command.		

Table 7. Status Register Definitions

Bit Number	Symbol	Description
Status Register 0		
7,6	IC	Interrupt Code. 00 — Normal termination of command. The specified command was properly executed and completed without error. 01 — Abnormal termination of command. Command execution was started but could not be successfully completed. 10 — Invalid command. The requested command could not be executed. 11 — Abnormal termination. During command execution, the disk drive ready signal changed state.
5	SE	Seek End. This flag is set (high) when the FDC has completed the Seek command and the read/write head is positioned over the correct cylinder.
4	EC	Equipment Check Error. This flag is set (high) if a fault signal is received from the disk drive or if the track 0 signal fails to become active after 77 step pulses (Recalibrate command).
3	NR	Not Ready Error. This flag is set if a read or write command is issued and either the drive is not ready or the command specifies side 1 (head 1) of a single-sided disk.
2	H	Head Address. The head address at the time of the interrupt.
1,0	DS1,DS0	Drive Select. The number of the drive selected at the time of the interrupt.

Table 7. Status Register Definitions (continued)

Bit Number	Symbol	Description
Status Register 1		
7	EN	End of Track Error. This flag is set if the FDC attempts to access a sector beyond the final sector of the track.
6		Not used. This bit is always low.
5	DE	Data Error. Set when the FDC detects a CRC error in either the ID field or the data field of a sector.
4	OR	Overrun Error. Set (during data transfers) if the FDC does not receive DMA or processor service within the specified time interval.
3		Not used. This bit is always low.
2	ND	Sector Not Found Error. This flag is set by any of the following conditions. a) The FDC cannot locate the sector specified in the Read Data, Read Deleted Data, or Scan command. b) The FDC cannot locate the starting sector specified in the Read Track command. c) The FDC cannot read the ID field without error during a Read ID command.
1	NW	Write Protect Error. This flag is set if the FDC detects a write protect signal from the disk drive during the execution of a Write Data, Write Deleted Data, or Format Track command.
0	MA	Missing Address Mark Error. This flag is set by either of the following conditions: a) The FDC cannot detect the ID address mark on the specified track (after two occurrences of the physical index mark). b) The FDC cannot detect the data address mark or deleted data address mark on the specified track. (See also the MD bit of Status Register 2.)
Status Register 2		
7		Not used. This bit is always low.
6	CM	Control Mark. This flag is set when the FDC encounters one of the following conditions: a) A deleted data address mark during the execution of a Read Data or Scan command. b) A data address mark during the execution of a Read Deleted Data command.
5	DD	Data Error. Set (high) when the FDC detects a CRC error in a sector data field. This flag is not set when a CRC error is detected in the ID field.
4	WC	Cylinder Address Error. Set when the cylinder address from the disk sector ID field is different from the current cylinder address maintained within the FDC.
3	SH	Scan Hit. Set during the execution of the Scan command if the scan condition is satisfied.
2	SN	Scan Not Satisfied. Set during execution of the Scan command if the FDC cannot locate a sector on the specified cylinder that satisfies the scan condition.
1	BC	Bad Track Error. Set when the cylinder address from the disk sector ID field is FF hexadecimal and this cylinder address is different from the current cylinder address maintained within the FDC. This all "ones" cylinder number indicates a bad track (one containing hard errors) according to the IBM soft-sectored format specifications.
0	MD	Missing Data Address Mark Error. Set if the FDC cannot detect a data address mark or deleted data address mark on the specified track.

Table 7. Status Register Definitions (continued)

Bit Number	Symbol	Description
Status Register 3		
7	FT	Fault. This flag indicates the status of the fault signal from the selected disk drive.
6	WP	Write Protected. This flag indicates the status of the write protect signal from the selected disk drive.
5	RDY	Ready. This flag indicates the status of the ready signal from the selected disk drive.
4	TO	Track 0. This flag indicates the status of the track 0 signal from the selected disk drive.
3	TS	Two-Sided. This flag indicates the status of the two-sided signal from the selected disk drive.
2	H	Head Address. This flag indicates the status of the side select signal for the currently selected disk drive.
1,0	DS1,DS0	Drive Select. Indicates the currently selected disk drive number.

In the non-DMA mode, transfer requests are indicated by activation of both the INT output signal and the RQM flag (bit 7) in the Main Status Register. INT can be used for interrupt-driven systems and RQM can be used for polled systems. The system processor must respond to the transfer request by reading data from (activating $\overline{RD}$), or writing data to (activating $\overline{WR}$), the FDC. This response removes the transfer request (INT and RQM are set inactive). After completing the last transfer, the 8272 activates the INT output to indicate the beginning of the result phase. In the non-DMA mode, the processor must activate the TC signal to the FDC (normally by means of an I/O port) after the transfer request for the last data byte has been received (by the processor) and before the appropriate data byte has been read from (or written to) the FDC.

In either mode of operation (DMA or non-DMA), the execution phase ends when a terminal count signal is sensed or when the last sector on a track (the EOT parameter—Table 5) has been read or written. In addition, if the disk drive is in a "not ready" state at the beginning of the execution phase, the "not ready" flag (bit 3 in Status Register 0) is set (high) and the command is terminated.

If a fault signal is received from the disk drive at the end of a write operation (Write Data, Write Deleted Data, or Format), the FDC sets the "equipment check" flag (bit 4 in Status Register 0), and terminates the command after setting the interrupt code (bits 7 and 6 of Status Register 0) to "01" (bit 7 low, bit 6 high).

Multi-sector and Multi-track Transfers

During disk read/write transfers (Read Data, Write Data, Read Deleted Data, and Write Deleted Data), the FDC will continue to transfer data from sequential sectors until the TC input is sensed. In the DMA mode, the TC input is normally connected to the TC/EOP (terminal count) output of the DMA controller. In the non-DMA mode, the processor directly controls the FDC TC input as previously described. Once the TC input is received, the FDC stops requesting data transfers (from the system processor or DMA controller). The FDC, however, continues to read data from, or write data to, the floppy disk until the end of the current disk sector. During a disk read operation, the data read from the disk (after reception of the TC input) is discarded, but the data CRC is checked for errors; during a disk write operation, the remainder of the sector is filled with all-zero bytes.

If the TC signal is not received before the last byte of the current sector has been transferred to/from the system, the FDC increments the sector number by one and initiates a read or write command for this new disk sector.

The FDC is also designed to operate in a multi-track mode for dual-sided disks. In the multi-track mode (specified by means of the MT flag in the command byte—Table 5) the FDC will automatically increment the head address (from 0 to 1) when the last sector (on the track under head 0) has been read or written. Reading or writing is then continued on the first sector (sector 1) of head 1.

Drive Status Polling

After the power-on reset, the 8272 automatically enters a drive status polling mode. If a change in drive status is detected (all drives are assumed to be "not ready" at power-on), an interrupt is generated. The 8272 continues this status polling between command executions (and between step pulses in the Seek command). In this manner, the 8272 automatically notifies the system processor when a floppy disk is inserted, removed, or changed by the operator.

Command Details

During the command phase, the Main Status Register must be polled by the CPU before each byte is written into the Data Register. The DI0 (bit 6) and RQM (bit 7) flags in the Main Status Register must be low and high, respectively, before each byte of the command may be written into the 8272. The beginning of the execution phase for any of these commands will cause DI0 to be set high and RQM to be set low.

The following paragraphs describe the fifteen FDC commands in detail.

Specify

The Specify command is used prior to performing any disk operations (including the formatting of a new disk) to define drive/FDC operating characteristics. The Specify command parameters set the values for three internal timers:

1. Head Load Time (HLT) — This seven-bit value defines the time interval that the FDC waits after loading the head before initiating a read or write operation. This timer is programmable from 2 to 254 milliseconds in increments of 2 ms.
2. Head Unload Time (HUT) — This four-bit value defines the time from the end of the execution phase (of a read or write command) until the head is unloaded. This timer is programmable from 16 to 240 milliseconds in increments of 16 ms. If the processor issues another command before the head unloads, the head will remain loaded and the head load wait will be eliminated.
3. Step Rate Time (SRT) — This four-bit value defines the time interval between step pulses issued by the FDC (track-to-track access time). This timer is programmable from 1 to 16 milliseconds in increments of 1 ms.

The time intervals mentioned above are a direct function of the FDC clock (CLK on pin 19). Times indicated above are for an 8 MHz clock.

The Specify command also indicates the choice of DMA or non-DMA operation (by means of the ND bit). When this bit is high the non-DMA mode is selected; when ND is low, the DMA mode is selected.

Sense Drive Status

This command may be used by the processor whenever it wishes to obtain the status of the disk drives. Status Register 3 (returned during the result phase) contains the drive status information as described in Table 7.

Sense Interrupt Status

An interrupt signal is generated by the FDC when one or more of the following events occurs:

1. The FDC enters the result phase for:
   a. Read Data command
   b. Read Track command
   c. Read ID command
   d. Read Deleted Data command
   e. Write Data command
   f. Format Track command
   g. Write Deleted Data command
   h. Scan commands
2. The ready signal from one of the disk drives changes state.
3. A Seek or Recalibrate command completes operation.
4. The FDC requires a data transfer during the execution phase of a command in the non-DMA mode.

Interrupts caused by reasons (1) and (4) above occur during normal command operations and are easily discernible by the processor. However, interrupts caused by reasons (2) and (3) above are uniquely identified with the aid of the Sense Interrupt Status command. This command, when issued, resets the interrupt signal and by means of bits 5, 6, and 7 of Status Register 0 (returned during the result phase) identifies the cause of the interrupt (see Table 8).

Table 8. Interrupt Codes

Seek End Bit 5	Interrupt Code		Cause
	Bit 6	Bit 7	
0	1	1	Ready Line changed state, either polarity
1	0	0	Normal Termination of Seek or Recalibrate Command
1	1	0	Abnormal Termination of Seek or Recalibrate Command

Neither the Seek nor the Recalibrate command has a result phase. Therefore, it is mandatory to use the Sense Interrupt Status Command after these commands to effectively terminate them and to provide verification of the disk head position.

When an interrupt is received by the processor, the FDC busy flag (bit 4) and the non-DMA flag (bit 5) may be used to distinguish the above interrupt causes:

bit 5	bit 4	
0	0	Asynchronous event-(2) or (3) above
0	1	Result phase-(1) above
1	1	Data transfer required-(4) above

A single interrupt request to the processor may, in fact, be caused by more than one of the above events. The processor should continue to issue Sense Interrupt Status commands (and service the resulting conditions) until an invalid command code is received. In this manner, all "hidden" interrupts are serviced.

Seek

The Seek command causes the drive's read/write head to be positioned over the specified cylinder. The FDC determines the difference between the current cylinder address and the desired (specified) address, and issues the appropriate number of step pulses. If the desired cylinder address is larger than the current address, the direction signal (LCT/DIR, pin 38) is set high (step-in); the direction signal is set low (step-out) if the desired cylinder address is less than the current address. No head movement occurs (no step pulses are issued) if the desired cylinder is the same as the current cylinder.

The rate at which step pulses are issued is controlled by the step rate time (SRT) in the Specify command. After each step pulse is issued, the desired cylinder address is compared against the current cylinder address. When the cylinder addresses are equal, the "seek end" flag (bit 5 in Status Register 0) is set (high) and the command is terminated. If the disk drive becomes "not ready" during the seek operation, the "not ready" flag (in Status Register 0) is set (high) and the command is terminated.

During the command phase of the Seek operation the FDC is in the FDC busy state, but during the execution phase it is in the non-busy state. While the FDC is in the non-busy state, another Seek command may be issued. In this manner parallel seek operations may be in operation on up to four floppy disk drives at once. The Main Status Register contains a flag for each drive (Table 4) that indicates whether the associated drive is currently operating in the seek mode. When a drive has completed a seek operation, the FDC generates an interrupt. In response to this interrupt, the system software must issue a Sense Interrupt Status command. During the result phase of this command, Status Register 0 (containing the drive number in bits 0 and 1) is read by the processor.

Recalibrate

This command causes the read/write head of the disk drive to retract to the track 0 position. The FDC clears the contents of its internal cylinder counter, and checks the status of the track 0 signal from the disk drive. As long as the track 0 signal is low, the direction signal remains high and step pulses are issued. When the track 0 signal goes high, the seek end flag (in Status Register 0) is set (high) and the command is terminated. If the track 0 signal is still low after 77 step pulses have been issued, the seek end and equipment check flags (in Status Register 0) are both set and the Recalibrate command is terminated.

Recalibrate commands for multiple drives can be overlapped in the same manner that Seek commands are overlapped.

Format Track

The Format Track command formats or "initializes" a track on a floppy disk by writing the ID field, gaps, and address marks for each sector. Before issuing the Format command, the Seek command must be used to position the read/write head over the correct cylinder. In addition, a table of ID field values (cylinder, head, and sector addresses and sector length code) must be prepared before the command is executed. During command execution, the FDC accesses the table and, using the values supplied, writes each sector on the track. The ID field address mark originates from the FDC and is written automatically as the first byte of each sector's ID field. The cylinder, head, and sector addresses are taken, in order, from the table. The ID field CRC character (derived from the data written in the first five bytes) is written as the last two bytes of the ID field. Gaps are written automatically by the FDC, with the length of the variable gap determined by one of the Format command parameters.

The data field address mark is generated by the FDC and is written automatically as the first byte of the data field. The data pattern specified in the command phase is written into each data byte of each sector. A CRC character is derived from the data address mark and the data written in the sector's data field. The two CRC bytes are appended to the last data byte.

The formatting of a track begins at the physical index mark. As previously mentioned, the order of sector assignment is taken directly from the formatting table. Four entries are required for each sector: a cylinder address, a head address, a sector address, and a sector length code. The cylinder address in the ID field should be equal to the cylinder address of the track currently being formatted.

The sector addresses must be unique (no two equal). The order of the sector entries in the table is the sequence in which sector numbers appear on the track when it is formatted. The number of entry sets (cylinder, head, and sector address and sector length code) must equal the number of sectors allocated to the track (specified in the command phase).

Since the sector address is supplied, in order, for each sector, tracks can be formatted sequentially (the first sector following the index mark is assigned sector address 1, the adjacent sector is assigned sector address 2, and so on) or sector numbers can be interleaved (see section 3) on a track.

Table 9 lists recommended gap sizes and sectors/track for various sector sizes.

Read Data

Nine (9) bytes are required to complete the command phase specification for the Read Data command. During the execution phase, the FDC loads the head (if it is in the unloaded state), waits the specified head load time (defined in the Specify command), and begins reading ID address marks and ID fields. When the requested sector address compares with the sector address read from the disk, the FDC outputs data (from the data field) byte-by-byte to the system. The Read Data command automatically operates in the multi-sector mode described earlier. In addition, multi-track operation may be specified by means of the MT command flag (Table 5). The amount of data that can be transferred with a single command to the FDC depends on the multi-track flag, the recording density flag, and the number of bytes per sector.

During the execution of read and write commands, the special sector size parameter (DTL) is used to temporarily alter the effective disk sector size. By setting the sector size code (N) to zero, DTL may be used to specify a sector size from 1 to 256 bytes in length. If the actual sector (on the disk) is larger than DTL specifies, only the number of bytes specified by the DTL parameter are passed to the system; the remainder of the actual disk sector is not transferred (although the data is checked for CRC errors). Multi-sector read operations are performed in the same manner as they are when the sector size code is non-zero. (The N and DTL parameters are always present in the command sequence. DTL should be set to FF hexadecimal when N is not zero.)

If the FDC detects the physical index mark twice without finding the requested sector, the FDC sets the "sector not found error" flag (bit 2 in Status Register 1) and terminates the Read Data command. The interrupt code (bits 7 and 6 of Status Register 0) is set to "01." Note that the FDC searches for each sector in a multi-sector operation. Therefore, a "sector not found" error may occur after successful transfer of one or more preceding sectors. This error could occur if a particular sector number was not included when the track was first formatted or if a hard error on the disk has invalidated a sector ID field.

After reading the ID field and data field in each sector, the FDC checks the CRC bytes. If a read error is detected (incorrect CRC in the ID field), the FDC sets the "data error" flag in Status Register 1; if a CRC error occurs in the data field, the FDC sets the "data error" flag in Status Register 2. In either error condition, the FDC terminates the Read Data command. The interrupt code (bits 7 and 6 in Status Register 0) is set to "01."

If the FDC reads a deleted data address mark from the disk, and the skip flag (specified during the command phase) is not set, the FDC sets the "control mark" flag (bit 6 in Status Register 2) and terminates the Read Data command (after reading all the data in the sector). If the skip flag is set, the FDC skips the sector with the deleted data address mark and reads the next sector. Thus, the skip flag may be used to cause the FDC to ignore deleted data sectors during a multi-sector read operation.

During disk data transfers between the FDC and the system, the FDC must be serviced by the system (processor or DMA controller) every 27 μs in the FM mode, and every 13 μs in the MFM mode. If the FDC is not

Table 9. Sector Size Relationships

Format	Sector Size	N Sector Size Code	SC Sectors/ Track	GPL[1] Gap 3 Length	GPL[2] Gap 3 Length	Remarks
FM Mode	128 bytes/Sector	00	$1A_{(16)}$	$07_{(16)}$	$1B_{(16)}$	IBM Diskette 1
	256	01	$0F_{(16)}$	$0E_{(16)}$	$2A_{(16)}$	IBM Diskette 2
	512	02	08	$1B_{(16)}$	$3A_{(16)}$	
MFM Mode	256	01	$1A_{(16)}$	$0E_{(16)}$	$36_{(16)}$	IBM Diskette 2D
	512	02	$0F_{(16)}$	$1B_{(16)}$	$54_{(16)}$	
	1024	03	08	$35_{(16)}$	$74_{(16)}$	IBM Diskette 2D

Notes: 1. Suggested values of GPL in Read or Write commands to avoid splice point between data field and ID field of contiguous sectors.
2. Suggested values of GPL in Format command.

serviced within this interval, the "overrun error" flag (bit 4 in Status Register 1) is set and the Read Data command is terminated.

If the processor terminates a read (or write) operation in the FDC, the ID information in the result phase is dependent upon the state of the multi-track flag and end of track byte. Table 11 shows the values for C, H, R, and N, when the processor terminates the command.

Write Data

Nine (9) bytes are required to complete the command phase specification for the Write Data command. During the execution phase the FDC loads the head (if it is in the unloaded state), waits the specified head load time (defined by the Specify command), and begins reading sector ID fields. When the requested sector address compares with the sector address read from the disk, the FDC reads data from the processor one byte at a time via the data bus and outputs the data to the data field of that sector. The CRC is computed on this data and two CRC bytes are written at the end of the data field.

The FDC reads the ID field of each sector and checks the CRC bytes. If the FDC detects a read error (incorrect CRC) in one of the ID fields, it sets the "data error" flag (bit 5 in Status Register 1) and terminates the Write Data command. The interrupt code (bits 7 and 6 in Status Register 0) is set to "01."

The Write Data command operates in much the same manner as the Read Data command. The following items are the same; refer to the Read Data command for details:

- Multi-sector and Multi-track operation
- Data transfer capacity
- "End of track error" flag
- "Sector not found error" flag
- "Data error" flag
- Head unload time interval
- ID information when the processor terminates the command (see Table 11)
- Definition of DTL when $N=0$ and when $N\neq 0$

During the Write Data execution phase, data transfers between the processor and FDC must occur every 31 μs in the FM mode, and every 15 μs in the MFM mode. If the time interval between data transfers is longer than this, the FDC sets the "overrun error" flag (bit 4 in Status Register 1) and terminates the Write Data command.

Read Deleted Data

This command operates in almost the same manner as the Read Data command operates. The only difference involves the treatment of the data address mark and the skip flag. When the FDC detects a data address mark at the beginning of a data field (and the skip flag is not set), the FDC reads all the data in the sector, sets the "control mark" flag (bit 6 in Status Register 2), and terminates the command. If the skip flag is set, the FDC skips the sector with the data address mark and continues reading at the next sector. Thus, the skip flag may be used to cause the FDC to read only deleted data sectors during a multi-sector read operation.

Write Deleted Data

This command operates in the same manner as the Write Data command operates except that a deleted data address mark is written at the beginning of the data field instead of the normal data address mark. This command is used to mark a bad sector (containing a hard error) on the floppy disk.

Read Track

The Read Track command is similar to the Read Data command except that the entire data field is read continuously from each of the sectors of a track. Immediately after encountering the physical index mark, the FDC starts reading all data fields on the track as continuous blocks of data. If the FDC finds an error in the ID field or data field CRC check bytes, it continues to read data from the track. The FDC compares the ID information read from each sector with the values specified during the command phase. If the specified ID field information is not found on the track, the "sector not found error" flag (in Status Register 1) is set. Multi-track and skip operations are not allowed with this command.

This command terminates when the last sector on the track has been read. (The number of sectors on the track is specified by the end of track parameter byte during the command phase.) If the FDC does not find an ID address mark on the disk after it encounters the physical index mark for the second time, it sets the "missing address mark error" flag (bit 0 in Status Register 1) and terminates the command. The interrupt code (bits 7 and 6 of Status Register 0) is set to "01."

Read ID

The Read ID command transfers (reads) the first correct ID field from the current disk track (following the physical index mark) to the processor. If no correct ID address mark is found on the track, the "missing address mark error" flag is set (bit 0 in Status Register 1). If no data mark is found on the track, the "sector not found error" flag is also set (bit 2 in Status Register 1). Either error condition causes the command to be terminated.

Scan Commands

The Scan commands allow the data being read from the disk to be compared against data supplied by the system (by the processor in non-DMA mode, and by the DMA controller in DMA mode). The FDC compares the data on a byte-by-byte basis, and searches for a sector of data that meets the conditions of "disk data equal to system data", "disk data less than or equal to system data", or "disk data greater than or equal to system data". Simple binary (ones complement) arithmetic is used for comparison (FF = largest number, 00 = smallest number). If, after a complete sector of data is compared, the conditions are not met, the sector number is incremented by the scan sector increment (specified in the command phase), and the scan operation is continued. The scan operation continues until one of the following conditions occurs; the conditions for scan are met (equal, low, or high), the last sector on the track is reached, or the terminal count signal is received.

If the conditions for scan are met, the FDC sets the "scan hit" flag (bit 3 in Status Register 2) and terminates the Scan command. If the conditions for scan are not met between the starting sector and the last sector on the track (specified in the command phase), the FDC sets the "scan not satisfied" flag (bit 2 in Status Register 2) and terminates the Scan command. The receipt of a terminal count signal from the processor or DMA controller during the scan operation will cause the FDC to complete the comparison of the particular byte which is in process, and to terminate the command. Table 10 shows the status of the "scan hit" and "scan

Table 10. Scan Status Codes

Command	Status Register 2		Comments
	Bit 2 = SN	Bit 3 = SH	
Scan Equal	0	1	$D_{FDD} = D_{Processor}$
	1	0	$D_{FDD} \neq D_{Processor}$
Scan Low or Equal	0	1	$D_{FDD} = D_{Processor}$
	0	0	$D_{FDD} < D_{Processor}$
	1	0	$D_{FDD} \nleq D_{Processor}$
Scan High or Equal	0	1	$D_{FDD} = D_{Processor}$
	0	0	$D_{FDD} > D_{Processor}$
	1	0	$D_{FDD} \ngeq D_{Processor}$

Table 11. ID Information When Processor Terminates Command

MT	EOT	Final Sector Transferred to Processor	ID Information at Result Phase			
			C	H	R	N
0	1A 0F 08	Sector 1 to 25 at Side 0 Sector 1 to 14 at Side 0 Sector 1 to 7 at Side 0	NC	NC	R + 1	NC
	1A 0F 08	Sector 26 at Side 0 Sector 15 at Side 0 Sector 8 at Side 0	C + 1	NC	R = 01	NC
	1A 0F 08	Sector 1 to 25 at Side 1 Sector 1 to 14 at Side 1 Sector 1 to 7 at Side 1	NC	NC	R + 1	NC
	1A 0F 08	Sector 26 at Side 1 Sector 15 at Side 1 Sector 8 at Side 1	C + 1	NC	R = 01	NC
1	1A 0F 08	Sector 1 to 25 at Side 0 Sector 1 to 14 at Side 0 Sector 1 to 7 at Side 0	NC	NC	R + 1	NC
	1A 0F 08	Sector 26 at Side 0 Sector 15 at Side 0 Sector 8 at Side 0	NC	$\overline{LSB}$	R = 01	NC
	1A 0F 08	Sector 1 to 25 at Side 1 Sector 1 to 14 at Side 1 Sector 1 to 7 at Side 1	NC	NC	R + 1	NC
	1A 0F 08	Sector 26 at Side 1 Sector 15 at Side 1 Sector 8 at Side 1	C + 1	$\overline{LSB}$	R = 01	NC

Notes: 1. NC (No Change): The same value as the one at the beginning of command execution.
2. $\overline{LSB}$ (Least Significant Bit): The least significant bit of H is complemented.

not satisfied" flags under various scan termination conditions.

If the FDC encounters a deleted data address mark in one of the sectors and the skip flag is low, it regards the sector as the last sector on the cylinder, sets the "control mark" flag (bit 6 in Status Register 2) and terminates the command. If the skip flag is high, the FDC skips the sector with the deleted address mark, and reads the next sector. In this case, the FDC also sets the "control mark" flag (bit 6 in Status Register 2) in order to show that a deleted sector had been encountered.

NOTE: During scan command execution, the last sector on the track must be read for the command to terminate properly. For example, if the scan sector increment is set to 2, the end of track parameter is set to 26, and the scan begins at sector 21, sectors 21, 23, and 25 will be scanned. The next sector, 27 will not be found on the track and an abnormal command termination will occur. The command would be completed in a normal manner if either a) the scan had started at sector 20 or b) the end of track parameter had been set to 25.

During the Scan command, data is supplied by the processor or DMA controller for comparison against the data read from the disk. In order to avoid having the "overrun error" flag set (bit 4 in Status Register 1), it is necessary to have the data available in less than 27 μs (FM Mode) or 13 μs (MFM Mode). If an overrun error occurs, the FDC terminates the command.

Invalid Commands

If an invalid (undefined) command is sent to the FDC, the FDC will terminate the command. No interrupt is generated by the 8272 during this condition. Bit 6 and bit 7 (DIO and RQM) in the Main Status Register are both set indicating to the processor that the 8272 is in the result phase and the contents of Status Register 0 must be read. When the processor reads Status Register 0 it will find an 80H code indicating that an invalid command was received.

A Sense Interrupt Status command must be sent after a Seek or Recalibrate interrupt; otherwise the FDC will consider the next command to be an invalid command. Also, when the last "hidden" interrupt has been serviced, further Sense Interrupt Status commands will result in invalid command codes.

In some applications the user may wish to use this command as a No-Op command to place the FDC in a stand-by or no operation state.

5. THE DATA SEPARATOR

As briefly discussed in section 2, LSI disk controllers such as the 8272 require external circuitry to generate a data window signal. This signal is used within the FDC to isolate the data bits contained within the READ DATA input signal from the disk drive. (The disk READ DATA signal is a composite signal constructed from both clock and data information.) After isolating the data bits from this input signal, the FDC assembles the data bits into 8-bit bytes for transfer to the system processor or memory.

Single Density

In single-density (FM) recording (Figure 3), the bit cell is 4 microseconds wide. Each bit cell contains a clock bit at the leading edge of the cell. The data bit (if present) is always located at the center of the cell. The job of data separation is relatively straightforward for single-density; simply generate a data window 2 μs wide starting 1 μs after each clock bit. Since every cell has a clock bit, a fixed window reference is available for every data bit and because the window is 2 μs wide, a slightly shifted data bit will still remain within the data window.

A single-density data separator with these specifications may be easily generated using a digital or analog one-shot triggered by the clock bit.

Double-Density

Double-density (MFM) bit cells are reduced to 2 μs (in order to double the disk data storage capacity). Clock bits are inserted into the data stream only if data bits are not present in both the current and preceding bit cells (Figure 3). The data bit (if present) still occurs at the center of the bit cell and the clock bit (if present) still occurs at the leading edge of the bit cell.

MFM data separation has two problems. First, only some bit cells contain a clock bit. In this manner, MFM encoding loses the fixed bit cell reference pulse present in FM encoding. Second, the bit cell for MFM is one-half the size of the bit cell for FM. This shorter bit cell means that MFM cannot tolerate as large a playback data-shift (as FM can tolerate) without errors.

Since most playback data-shift is predictable, the FDC can precompensate the write data stream so that data/clock pulses will be correctly positioned for subsequent playback. This function is completely controlled by the FDC and is only required for MFM recording. During write operations, the FDC specifies an early, normal, or late bit positioning. This timing information is specified with respect to the FDC write clock. Early and late timing is typically 125 ns to 250 ns before or after the write clock transition (depending on disk drive requirements).

The data separator circuitry for double-density recording must continuously analyze the total READ DATA stream, synchronizing its operation (window generation) with the actual clock/data bits of the data stream. The data separation circuit must track the disk input data frequency very closely—unpredictable bit shifts leave less than 50 ns margin to the window edges.

Phase-Locked Loop

Only an analog phase-locked loop (PLL) can provide the reliability required for a double-density data separation circuit. (A phase-locked loop is an electronic circuit that constantly analyzes the frequency of an input signal and locks another oscillator to that frequency.) Using analog PLL techniques, a data separator can be designed with ±1 ns resolution (this would require a 100 MHz clock in a digital phase-locked loop). The analog PLL determines the clock and data bit positions by sampling each bit in the serial data stream. The phase relationship between a data bit and the PLL generated data window is constantly fed back to adjust the position of the data window, enabling the PLL to track input data frequency changes, and thereby reliably read previously recorded data from a floppy disk.

PLL Design

A block diagram of the phase-locked loop described in this application note is shown in Figure 7. Basically, the phase-locked loop operates by comparing the frequency of the input data (from the disk drive) against the frequency of a local oscillator. The difference of these frequencies is used to increase or decrease the frequency of the local oscillator in order to bring its frequency closer to that of the input. The PLL synchronizes the local oscillator to the frequency of the input during the all "zeroes" synchronization field on the floppy disk (immediately preceding both the ID field and the data field).

The PLL consists of nine ICs and is located on page 3 of the schematics in the Appendix. The 8272 VCO output essentially turns the PLL circuitry on and off. When the PLL is off, it "idles" at its center frequency. The VCO output turns the PLL on only when valid data is being received from the disk drive. The VCO turns the PLL on after the read/write head has been loaded and the head load time has elapsed. The PLL is turned off in the gap between the ID field and the data field and in the gap after the data field (before the next sector ID field). The GPL parameter in the FDC read and write commands specifies the elapsed time (number of data bytes) that the PLL is turned off in order to blank out discontinuities that appear in the gaps when the write current is turned on and off. The PLL operates with either MFM or FM input data. The MFM output from the FDC controls the PLL operation frequency.

The PLL consists of six functional blocks as follows:

1. Pulse Shaping — A 96LS02 senses a READ DATA pulse and provides a clean output signal to the FDC and to the PLL Phase Comparator and Frequency Discriminator circuitry.
2. Phase Comparator — The phase difference between the PLL oscillator and the READ DATA input is compared. Pump up (PU) and pump down (PD) error signals are derived from this phase difference and output to the filter. If there is no phase difference between the PLL oscillator and the READ DATA input, the PU and PD pulse widths are equal. If the READ DATA pulse occurs early, the PU duration is shorter than the PD duration. If the data pulse occurs late, the PU duration is longer than the PD duration.
3. Filter — This analog circuit filters the PU and PD pulses into an error voltage. This error voltage is buffered by an LM358 operational amplifier.

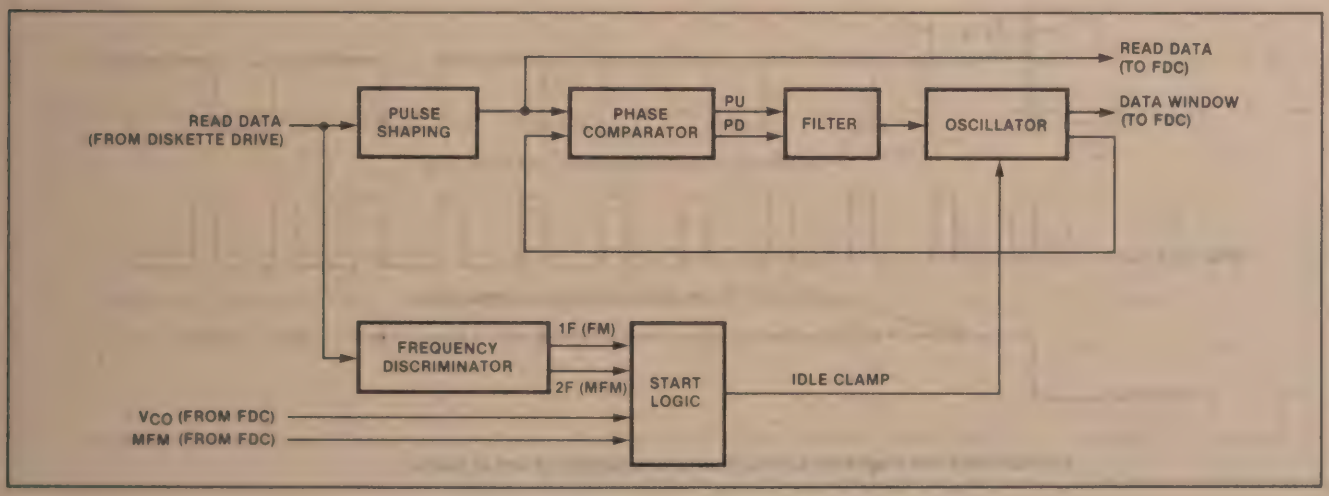


Figure 7. Phase-Locked Loop Data Separator

4. PLL Oscillator — This oscillator is composed of a 74LS393, 74LS74, and 96LS02. The oscillator frequency is controlled by the error voltage output by the filter. This oscillator also generates the data window signal to the FDC.
5. Frequency Discriminator — This logic tracks the READ DATA input from the disk drive and discriminates between the synchronization gap for FM recording (250 KHz) and the gap for MFM recording (500 KHz). Synchronization gaps immediately precede address marks.
6. Start Logic — The function of this logic is to clamp the PLL oscillator to its center frequency (2 MHz) until the FDC VCO signal is enabled and a valid data pattern is sensed by the frequency discriminator. The start logic (consisting of a 74LS393 and 74LS74) ensures that the PLL oscillator is started with zero phase error.

PLL Adjustments

The PLL must be initially adjusted to operate at its center frequency with the VCO output off and the adjustment jumper removed. The 5K trimpot should be adjusted until the frequency at the test point (Q output of the 96LS02) is 2 MHz. The jumper should then be replaced for normal operation.

PLL Design Details

The following paragraphs describe the operational and design details of the phase-locked loop data separator illustrated in the appendix. Note that the analog section is operated from a separately filtered +5V supply.

Initialization

As long as the 8272 maintains a low VCO signal, the data separator logic is "turned off". In this state, the PLL oscillator (96LS02) is not oscillating and therefore the 2XBR signal is constantly low. In addition, the pump up (PU) and pump down (PD) signals are inactive (PU low and PD high), the CNT8 signal is inactive (low), and the filter input voltage is held at 2.5 volts by two 1Mohm resistors between ground and +5 volts.

Floppy Disk Data

The data separator frequency discriminator, the input pulse shaping circuitry, and the start logic are always enabled and respond to rising edges of the READ DATA signal. The rising edge of every data bit from the disk drive triggers two pulse shaping one-shots. The first pulse shaper generates a stable and well-defined 200 ns read data pulse for input to the 8272 and other portions of the data separator logic. The second one-shot generates a 2.5 μs data pulse that is used for input data frequency discrimination.

The frequency discriminator operates as illustrated in Figure 8. The 2F output signal is active (high) during reception of valid MFM (double-density) sync fields on the disk while the 1F signal is active (high) during reception of valid FM (single-density) sync fields. A multiplexer (controlled by the 8272 MFM signal) selects the appropriate 1F or 2F signal depending on the programmed mode.

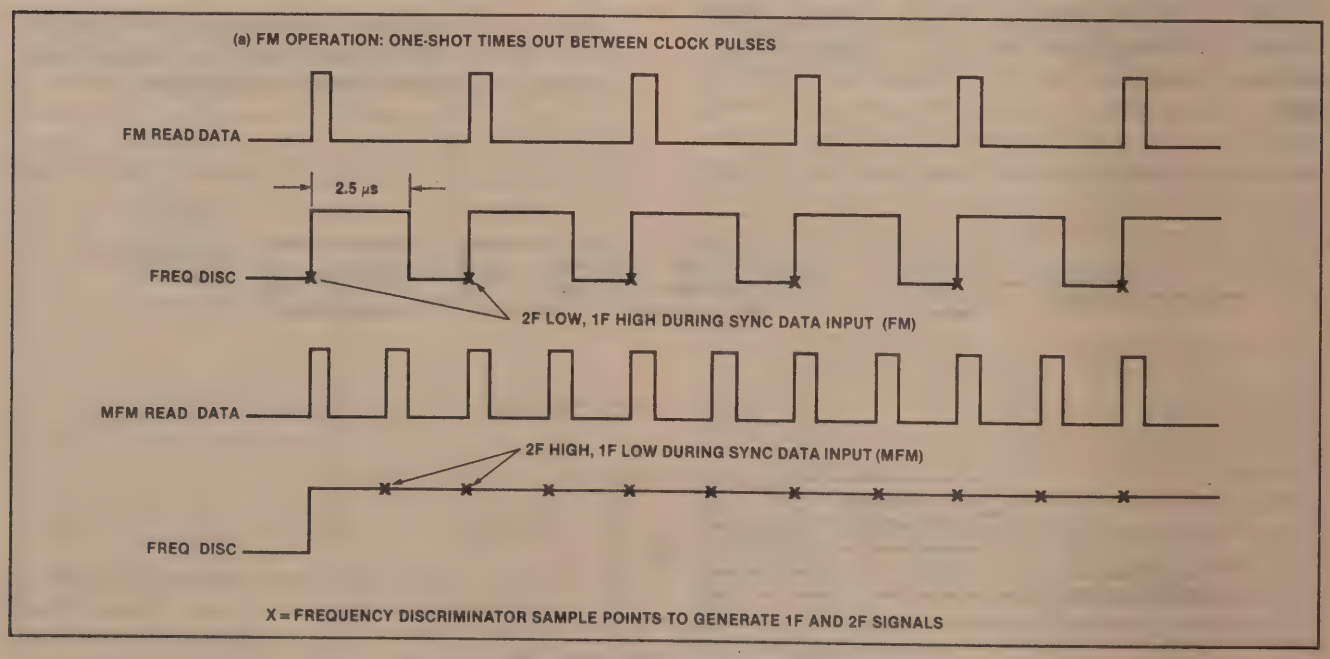


Figure 8. Input Data Frequency Discrimination

Startup

The data separator is designed to require reception of eight valid sync bits (one sync byte) before enabling the PLL oscillator and attempting to synchronize with the input data stream (see Figure 9). This delay ensures that the PLL will not erroneously synchronize outside a valid sync field in the data stream if the VCO signal is enabled slightly early. The sync bit counter is asynchronously reset by the CNTEN signal when valid sync data is not being received by the drive.

Once the VCO signal is active and eight sync bits have been counted, the CNT8 signal is enabled. This signal turns on the PLL oscillator. Note that this oscillator starts synchronously with the rising edge of the disk input data (because CNT8 is synchronous with the data rising edge) and the oscillator also starts at its center frequency of 2 MHz (because the LM348 filter input is held at its center voltage of approximately 2.5 volts). This frequency is divided by two and four to generate the 2XBR signal (1 MHz for MFM and 500 KHz for FM).

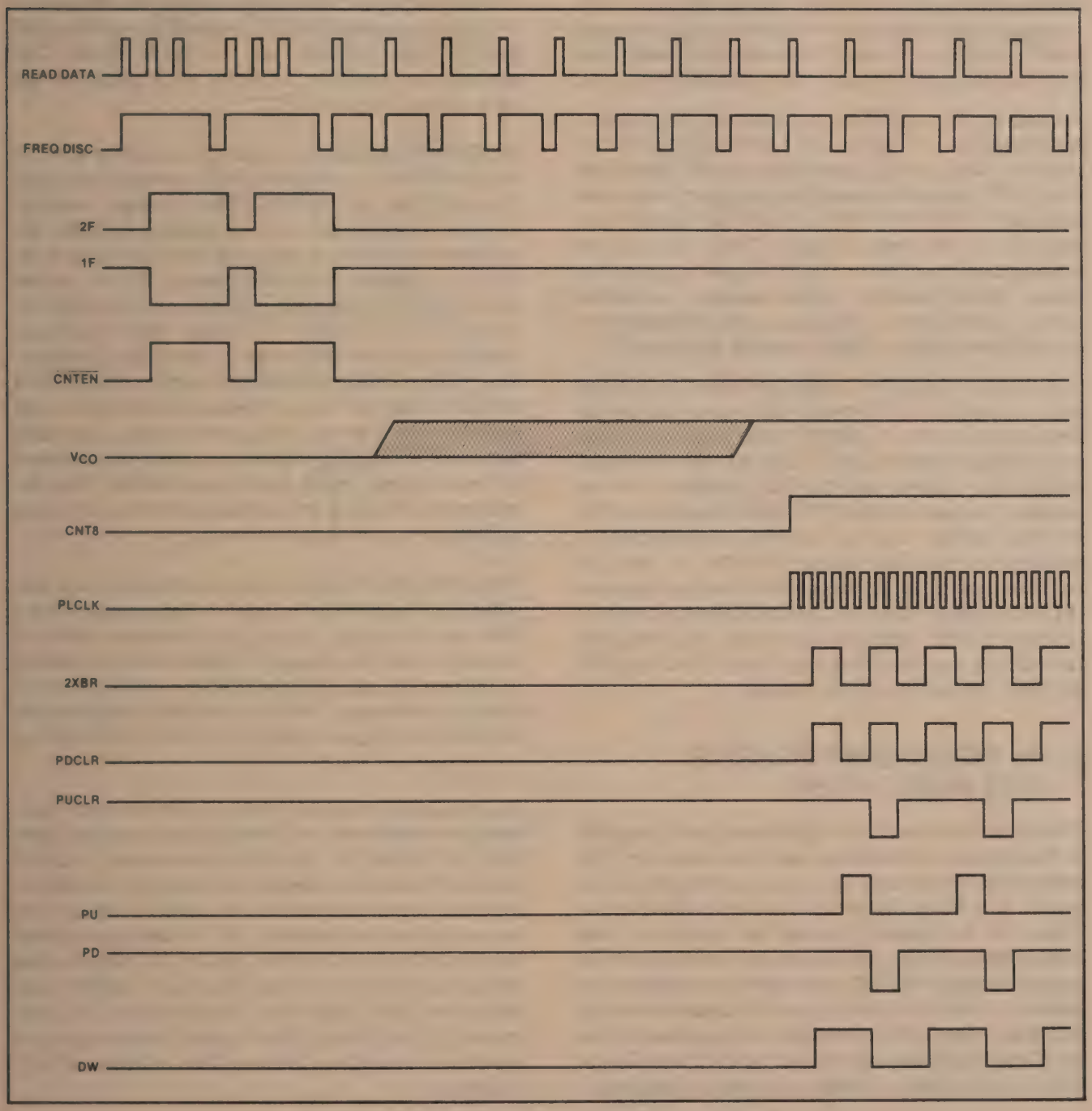


Figure 9. Typical Data Separator Startup Timing Diagram

PLL Synchronization

At this point, the PLL is enabled and begins to synchronize with the input data stream. This synchronization is accomplished very simply in the following manner. The pump up (PU) signal is enabled on the rising edge of the READ DATA from the disk drive. (When the PLL is synchronized with the data stream, this point will occur at the same time as the falling edge of the 2XBR signal as shown in Figure 9). The PU signal is turned off and the PD signal is activated on the next rising edge of the 2XBR clock. With this scheme, the difference between PU active time and the PD active time is equal to the difference between the input bit rate and the PLL clock rate. Thus, if PU is turned on longer than PD is on, the input bit rate is faster than the PLL clock.

As long as PU and PD are both inactive, no charge is transferred to or from the LM358 input holding capacitor, and the PLL output frequency is maintained (the LM358 operational amplifier has a very high input impedance). Whenever PU is turned on, current flows from the +5 volt supply through a 20K resistor into the holding capacitor. When the PD signal is turned on, current flows from the holding capacitor to ground through a 20K resistor. In this manner, both the pump up and pump down charging rates are balanced.

The change in capacitor charge (and therefore voltage) after a complete PU/PD cycle is proportional to the difference between the PU and PD pulse widths and is also proportional to the frequency difference between the incoming data stream and the PLL oscillator. As the capacitor voltage is raised (PU active longer than PD), the PLL oscillator time constant (RC of the 96LS02) is modified by the filter output (LM358) to raise the oscillator frequency. As the capacitor voltage is lowered (PD active longer than PD), the oscillator frequency is lowered. If both frequencies are equal, the voltage on the holding capacitor does not change, and the PLL oscillator frequency remains constant.

6. AN INTELLIGENT DISKETTE DATA BASE SYSTEM

The system described in this application note is designed to function as an intelligent data base controller. The schematics for this data base unit are presented in Appendix A; a block diagram of the unit is illustrated in Figure 10. As designed, the unit can access over four million bytes of mass storage on four floppy disk drives (using a single 8272 FDC); the system can easily be expanded to four FDC devices (and 16 megabytes of online disk storage). Three serial data links are also included. These data links may be used by CRT terminals or other microprocessor systems to access the data base.

Processor and Memory

A high-performance 8088 eight-bit microprocessor (operating at 5 MHz with no wait states) controls system operation. The 8088 was selected because of its memory addressing capabilities and its sophisticated string handling instructions. These features improve the speed of data base search operations. In addition, these capabilities allow the system to be easily upgraded with additional memory, disk drives, and if required, a bubble memory or winchester disk unit.

The schematics for the basic design provide 8K bytes of 2732A high-speed EPROM program storage and 8K bytes of disk directory and file buffer RAM. This memory can easily be expanded to 1 megabyte for performance upgrades.

An 8259A Programmable Interrupt Controller (PIC) is also included in the design to field interrupts from both the serial port and the FDC. This interrupt controller provides a large degree of programming flexibility for the implementation of data base functions in an asynchronous, demand driven environment. The PIC allows the system to accumulate asynchronous data base requests from all serial I/O ports while previously specified data base operations are currently in progress. This feature is made possible by the ability of the 8251A RXRDY signal to cause a processor interrupt. After receiving this interrupt, the processor can temporarily halt work on existing requests and enter the incoming information into a data base request buffer. Once the information has been entered into the buffer, the system can resume its previous processing.

In addition, the PIC permits some portions of data base requests to be processed in parallel. For example, once a disk record has been loaded into a memory buffer, a memory search can proceed in parallel with the loading of the next record. After the FDC completes the record transfer, the memory search will be interrupted and the processor can begin another disk transfer before resuming the memory search.

The bus structure of the system is split into three functional buffered units. A 20-bit address from the processor is latched by three-state transparent 74LS373 devices. When the processor is in control of the address and data busses, these devices are output enabled to the system buffered address bus. All I/O devices are placed directly on the local data bus. Finally, the memory data bus is isolated from the local data bus by an 8286 octal transceiver. The direction of this transceiver is determined by the Memory Read signal, while its output enable is activated by a Memory Read or Memory Write command.

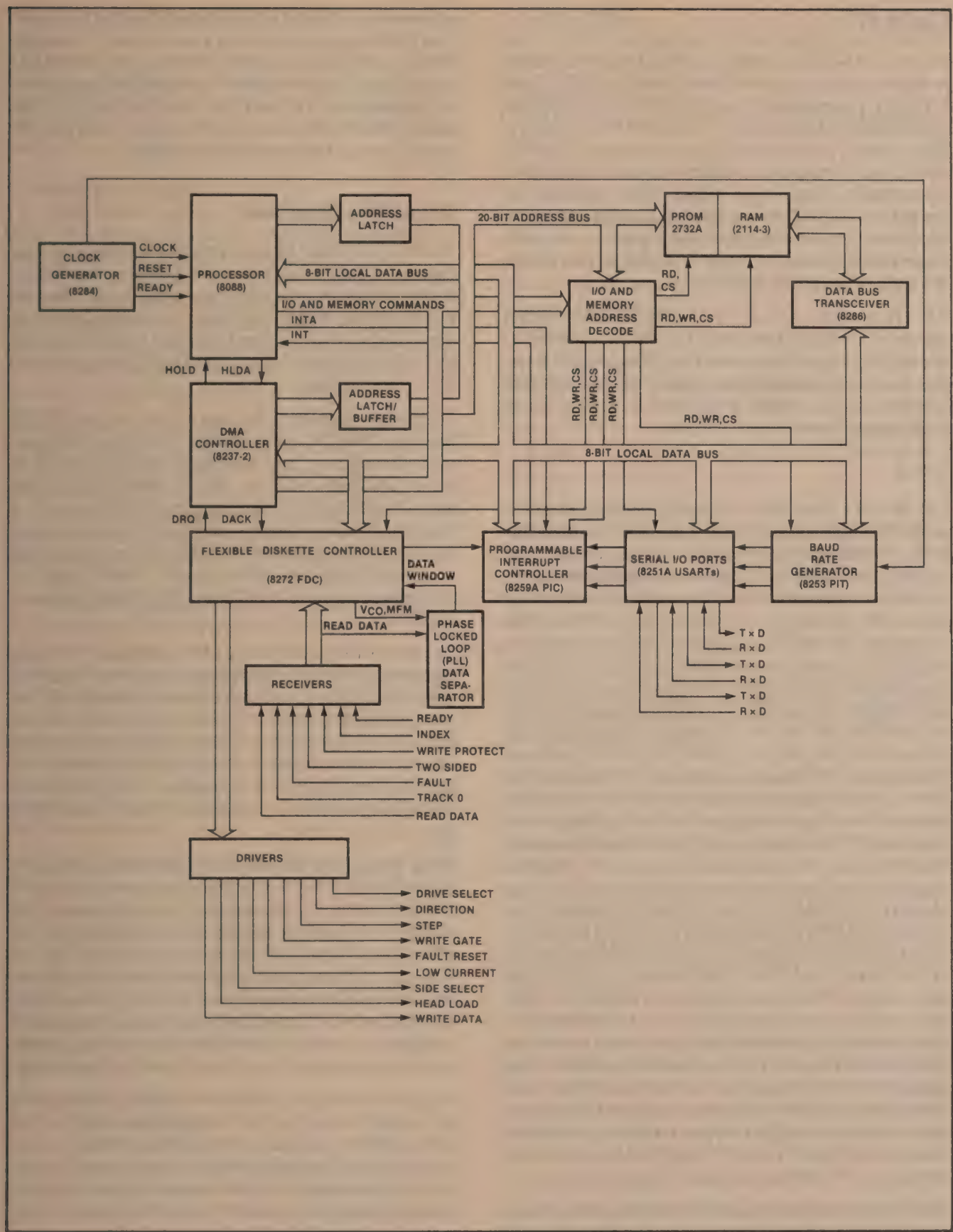


Figure 10. Intelligent Data Base Block Diagram

Serial I/O

The three RS-232-C compatible serial I/O ports operate at software-programmable baud rates to 19.2K. Each I/O port is controlled by an 8251A USART (Universal Synchronous/Asynchronous Receiver/Transmitter). Each USART is individually programmable for operation in many synchronous and asynchronous serial data transmission formats (including IBM Bi-sync). In operation, USART error detection circuits can check for parity, data overrun, and framing errors. An 8253 Programmable Interval Timer is employed to generate the baud rates for the serial I/O ports.

The Transmitter Ready and Receiver Ready output signals of the 8251As are routed to the interrupt inputs of the 8259A interrupt controller. These signals interrupt processor execution when a data byte is received by a USART and also when the USART is ready to accept another data byte for transmission.

DMA

The 8272 FDC interfaces to system memory by means of an 8237-2 high-speed DMA controller. Transfers between the disk controller and memory also operate with no wait states when 2114-3 (150 ns) or faster static RAM is used. In operation, the 8272 presents a DMA request to the 8237 for every byte of data to be transferred. This request causes the 8273 to present a HOLD request to the 8088. As soon as the 8088 is able to relinquish data/address bus control, the processor signals a HOLD acknowledge to the 8237. The 8237 then assumes control over the data and address busses. After latching the address for the DMA transfer, the 8237 generates simultaneous I/O Read and Memory Write commands (for a disk read) or simultaneous I/O Write and Memory Read commands (for a disk write). At the same time, the 8272 is selected as the I/O device by means of the DMA acknowledge signal from the 8237. After this single byte has been transferred between the FDC and memory, the DMA controller releases the data/address busses to the 8088 by deactivating the HOLD request. In a short period of time (13 μs for double-density and 27 μs for single-density) the FDC requests a subsequent data transfer. This transfer occurs in exactly the same manner as the previous transfer. After all data transfers have been completed (specified by the word count programmed into the 8237 before the FDC operation was initiated), the 8237 signals a terminal count (EOP pin). This terminal count signal informs the 8272 that the data transfer is complete. Upon reception of this terminal count signal, the 8272 halts DMA requests and initiates an "operation complete" interrupt.

Since the system is designed for 20-bit addressing, a four-bit DMA-address latch is included as a processor addressable I/O port. The processor writes the upper four DMA address bits before a data transfer. When the DMA controller assumes bus control, the contents of this latch are output enabled on the upper four bits of the address bus. The only restriction in the use of this address latch is that a single disk read or write transfer cannot cross a 64K memory boundary.

Disk Drive Interface

The 8272 FDC may be interfaced to a maximum of four eight-inch floppy disk drives. Both single- and double-density drives are accommodated using the data separation circuit described in section 5. In addition, single- or dual-sided disk drives may be used. The 8272 is driven by an 8 MHz crystal controller clock produced by an 8224 clock generator.

Drive select signals are decoded by means of a 74LS139 from the DS0, DS1 outputs of the FDC. The fault reset, step, low current, and direction outputs to the disk drives are generated from the FR/STEP, LCT/DIR, and RW/SEEK FDC output signals by means of a 74LS240. The other half of the 74LS240 functions as an input multiplexer for the disk write protect, two-sided, fault, and track zero status signals. These signals are multiplexed into the WP/TS and FLT/TRK0 inputs to the 8272.

The 8272 write clock (WR CLK) is generated by a ring counter/multiplexer combination. The write clock frequency is 1 MHz for MFM recording and 500 KHz for FM recording (selected by the MFM output of the 8272). The pulse width is a constant 250 ns. The write clock is constantly generated and input to the FDC (during both read and write operations). The FDC write enable output (WE) is transmitted directly to the write gate disk drive input.

Write data to the disk drive is preshifted (according to the PS0, PS1 FDC outputs) by the combination of a 74LS175 four-bit latch and a 74LS153 multiplexer. The amount of preshift is completely controlled within the 8272 FDC. Three cases are possible: the data may be written one clock cycle early, one clock cycle late, or with no preshift. The data preshift circuit is activated by the FDC only in the double-density mode. The preshift is required to cancel predictable playback data shifts when recorded data is later read from the floppy disk.

A single 50-conductor flat cable connects the board to the floppy disk drives. FDC outputs are driven by 7438 open collector high-current line-drivers. These drivers are resistively terminated on the last disk drive by means of a 150 ohm resistor to +5V. The line receivers are 7414 Schmitt triggered inverters with 150 ohm pull-up resistors on board.

7. SPECIAL CONSIDERATIONS

This section contains a quick review of key features and issues, most of which have been mentioned in other sections of this application note. Before designing with the 8272 FDC, it is advisable that the information in this section be completely understood.

1. Multi-Sector Transfers

The 8272 always operates in a multi-sector transfer mode. The 8272 continues to transfer data until the TC input is activated. In a DMA configuration, the TC input of the 8272 must always be connected to the EOP/TC output of the DMA controller. When multiple DMA channels are used on a single DMA controller, EOP must be gated with the select signal for the proper FDC. If the TC signal is not gated, a terminal count on another channel will abort FDC operation.

In a processor driven configuration with no DMA controller, the system must count the transfers and supply a TC signal to the FDC. In a DMA environment, ORing a programmable TC with the TC from the DMA controller is a convenient means of ensuring that the processor may always gain control of the FDC (even if the diskette system hangs up in an abnormal manner).

2. Processor Command/Result Phase Interface

In the command phase, the processor must write the exact number of parameters in the exact order shown in Table 5. During the result phase, the processor must read the complete result status. For example, the Format Track command requires six command bytes and presents seven result bytes. The 8272 will not accept a new command until all result bytes are read. Note that the number of command and result bytes varies from command-to-command. Command and result phases cannot be shortened.

During both the command and result phases, the Main Status Register must be read by the processor before each byte of information is read from, or written to, the FDC Data Register. Before each command byte is written, DIO (bit 6) must be low (indicating a data transfer from the processor) and RQM (bit 7) must be high (indicating that the FDC is ready for data). During the result phase, DIO must be high (indicating a data transfer to the processor) and RQM must also be high (indicating that data is ready for the processor).

NOTE: After the 8272 receives a command byte, the RQM flag may remain set for 12 microseconds (with an 8 MHz clock). Software should not attempt to read the Main Status Register before this time interval has elapsed; otherwise, the software will erroneously assume that the FDC is ready to accept the next byte.

3. Sector Sizes

The 8272 does not support 128 byte sectors in the MFM (double-density) mode.

4. Write Clock

The FDC Write Clock input (WR CLK) must be present at all times.

5. Reset

The FDC Reset input (RST) must be held active during power-on reset while the RD and WR inputs are active. If the reset input becomes inactive while RD and WR are still active, the 8272 enters the test mode. Once activated, the test mode can only be deactivated by a power-down condition.

6. Drive Status

The 8272 constantly polls (starting after the power-on reset) all drives for changes in the drive ready status. At power-on, the FDC assumes that all drives are not ready. If a drive application requires that the ready line be strapped active, the FDC will generate an interrupt immediately after power is applied.

7. Gap Length

Only the gap 3 size is software programmable. All other gap sizes are fixed. In addition, different gap 3 sizes must be specified in format, read, write, and scan commands. Refer to Section 3 and Table 9 for gap size recommendations.

8. Seek Command

The drive busy flag in the Main Status Register remains set after a Seek command is issued until the Sense Interrupt Status command is issued (following reception of the seek complete interrupt).

The FDC does not perform implied seeks. Before issuing data read or write commands, the read/write head must be positioned over the correct cylinder. If the head is not positioned correctly, a cylinder address error is generated.

After issuing a step pulse, the 8272 resumes drive status polling. For correct stepper operation in this mode, the stepper motor must be constantly enabled. (Most drives provide a jumper to permit the stepper motor to be constantly enabled.)

9. Step Rate

The 8272 can emit a step pulse that is one millisecond faster than the rate programmed by the SRT parameter in the Specify command. This action may cause subsequent sector not found errors. The step rate time should be programmed to be 1 ms longer than the step rate time required by the drive.

10. Cable Length

A cable length of less than 10 feet is recommended for drive interfacing.

11. Scan Commands

The current 8272 has several problems when using the scan commands. These commands should not be used at this time.

12. Interrupts

When the processor receives an interrupt from the FDC, the FDC may be reporting one of two distinct events:

a) The beginning of the result phase of a previously requested read, write, or scan command.

b) An asynchronous event such as a seek/recalibrate completion, an attention, an abnormal command termination, or an invalid command.

These two cases are distinguished by the FDC busy flag (bit 4) in the Main Status Register. If the FDC busy flag is high, the interrupt is of type (a). If the FDC busy flag is low, the interrupt was caused by an asynchronous event (b).

A single interrupt from the FDC may signal more than one of the above events. After receiving an interrupt, the processor must continue to issue Sense Interrupt Status commands (and service the resulting conditions) until an invalid command code is received. In this manner, all "hidden" interrupts are ferreted out and serviced.

13. Skip Flag (SK)

The skip flag is used during the execution of Read Data, Read Deleted Data, Read Track, and various Scan commands. This flag permits the FDC to skip unwanted sectors on a disk track.

When performing a Read Data, Read Track, or Scan command, a high SK flag indicates that the FDC is to skip over (not transfer) any sector containing a deleted data address mark. A low SK flag indicates that the FDC is to terminate the command (after reading all the data in the sector) when a deleted data address mark is encountered.

When performing a Read Deleted Data command, a high SK flag indicates that sectors containing normal data address marks are to be skipped. Note that this is just the opposite situation from that described in the last paragraph. When a data address mark is encountered during a Read Deleted Data command (and the SK flag is low), the FDC terminates the command after reading all the data in the sector.

14. Bad Track Maintenance

The 8272 does not internally maintain bad track information. The maintenance of this information must be performed by system software. As an example of typical bad track operation, assume that a media test determines that track 31 and track 66 of a given floppy disk are bad. When the disk is formatted for use, the system software formats physical track 0 as logical cylinder 0 (C=0 in the command phase parameters), physical track 1 as logical track 1 (C=1), and so on, until physical track 30 is formatted as logical cylinder 30 (C=30). Physical track 31 is bad and should be formatted as logical cylinder FF (indicating a bad track). Next, physical track 32 is formatted as logical cylinder 31, and so on, until physical track 67 is formatted as logical cylinder 64. Next, bad physical track 66 is formatted as logical cylinder FF (another bad track marker), and physical track 67 is formatted as logical cylinder 65. This formatting continues until the last physical track (77) is formatted as logical cylinder 75. Normally, after this formatting is complete, the bad track information is stored in a prespecified area on the floppy disk (typically in a sector on track 0) so that the system will be able to recreate the bad track information when the disk is removed from the drive and reinserted at some later time.

To illustrate how the system software performs a transfer operation disk with bad tracks, assume that the disk drive head is positioned at track 0 and the disk described above is loaded into the drive. If a command to read track 36 is issued by an application program, the system software translates this read command into a seek to physical track 37 (since there is one bad track between 0 and 36, namely 31) followed by a read of logical cylinder 36. Thus, the cylinder parameter C is set to 37 for the Seek command and 36 for the Read Sector command.

15. Head Load versus Head Settle Times

The 8272 does not permit separate specification of the head load time and the head settle time. When the Specify command is issued for a given disk drive, the proper value for the HLT parameter is the maximum of the head load time and the head settle time.

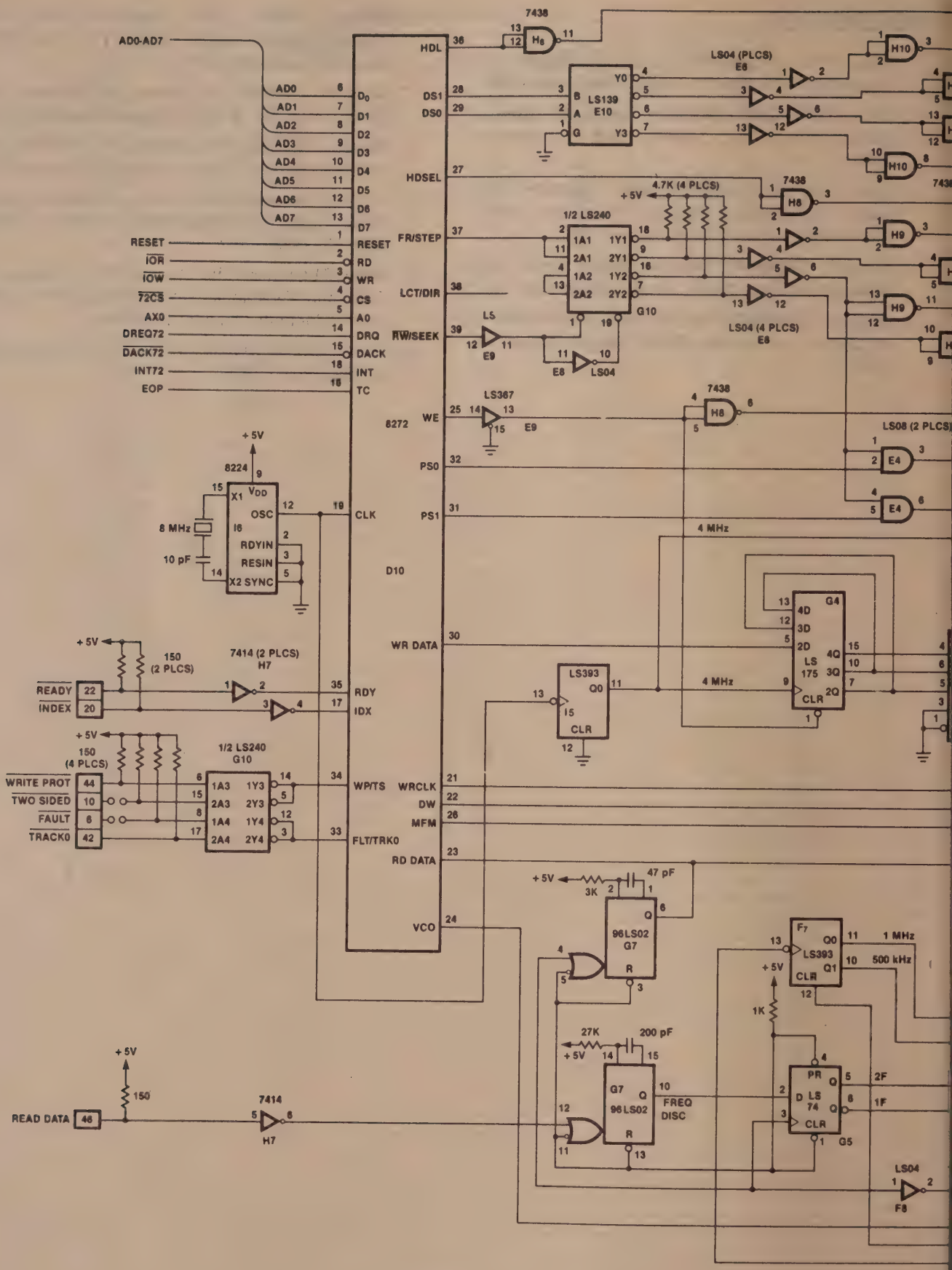
APPENDIX

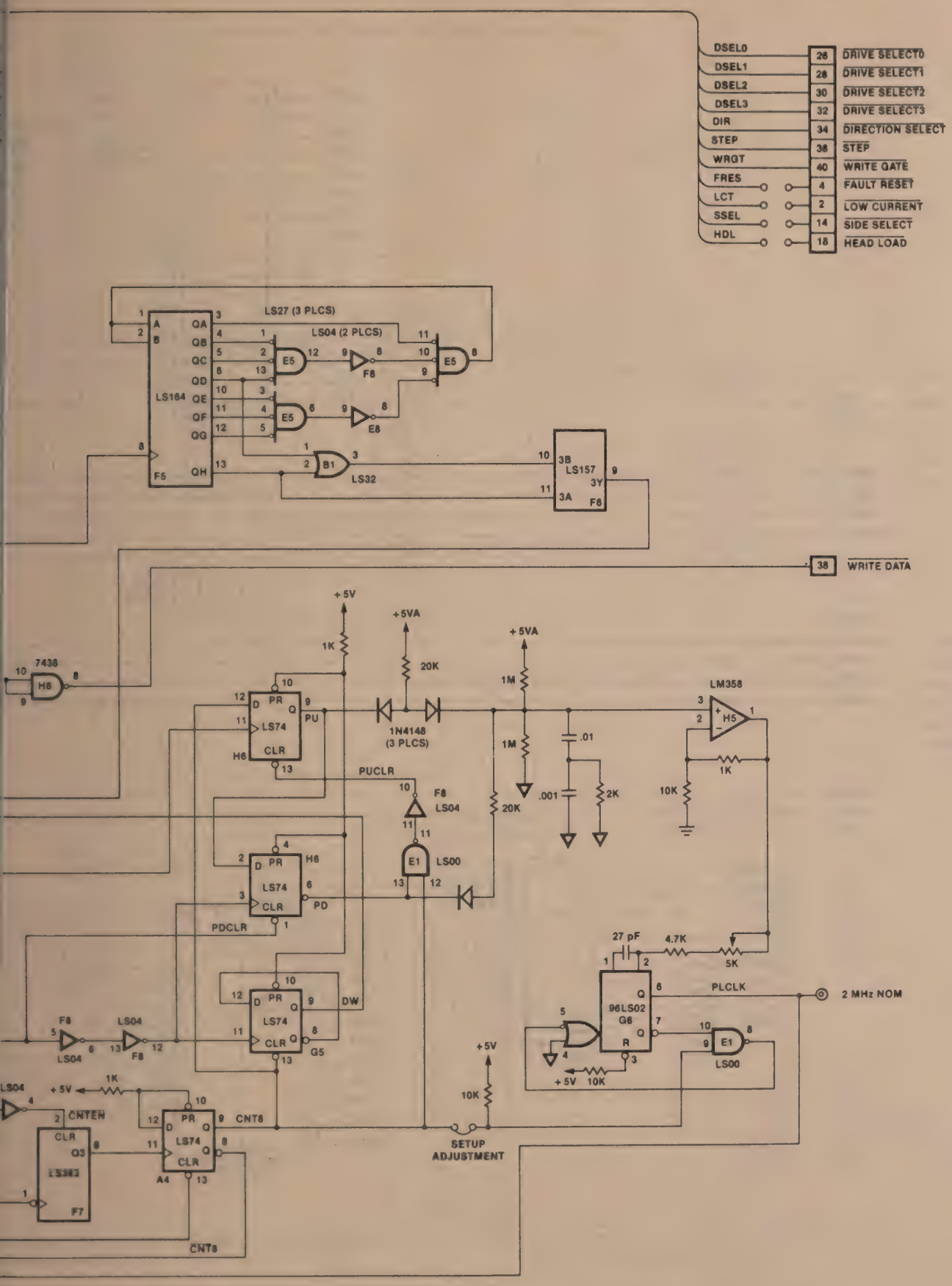
Power Distribution

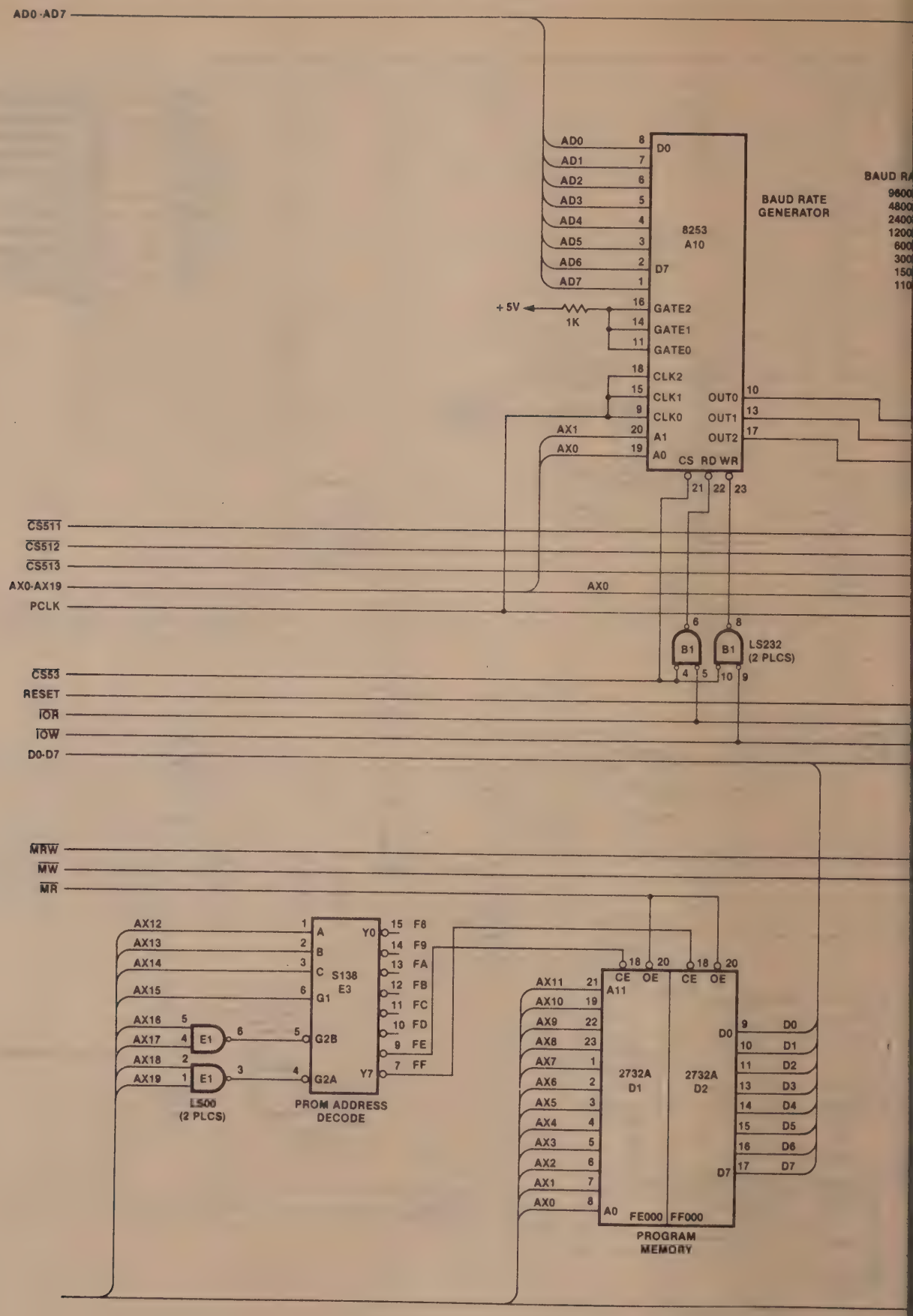
Part	Ref Desig	+5	GND	+12	−12
8088	A2	40	1,20		
8224	I6	9,16	8		
8237-2	A6	31	20		
8251A	A9,B9,C9	26	4		
8253-5	A10	24	12		
8259A	B10	28	14		
8272	D10	40	20		
8284	A1	18	9		
8286	B6,F4	20	10		
2114	F1,F2,G1,G2,H1,H2,I1,I2	18	9		
2732A	D1,D2	24	12		
74LS00	E1	14	7		
74LS04	B2,E6,E8,F8	14	7		
74LS27	E2,E5	14	7		
74LS32	B1	14	7		
74LS74	A4,G5,H6	14	7		
74LS138	F3	16	8		
74LS139	E10	16	8		
74LS153	I3	16	8		
74LS157	F6	16	8		
74LS164	F5	14	7		
74LS173	G3	16	8		
74LS175	G4	16	8		
74LS240	G10	20	10		
74LS257	D3	16	8		
74LS367	C3,E9	16	8		
74LS373	B4,C4,D4,C6	20	10		
74LS393	I5,F7	14	7		
74S08	E4	14	7		
74S138	D6,E3	16	8		
7414	H7	14	7		
7438	H8,H9,H10	14	7		
1488	H3		7	14	1
1489	H4	14	7		
96LS02	G7	16	8		
96LS02	G6			16	8
LM358	H5			8	4

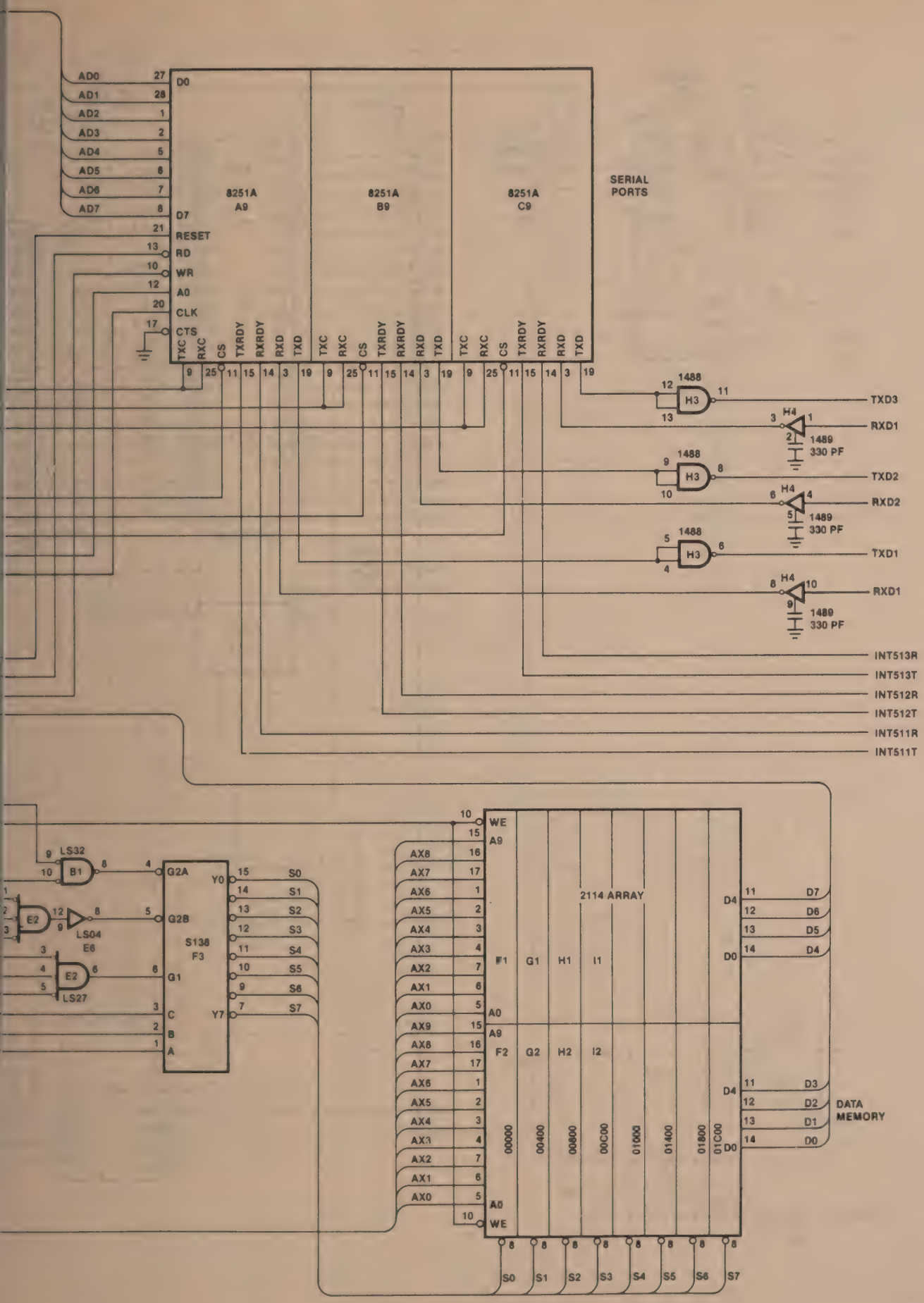
REFERENCES

1. Intel, "8272 Single/Double Density Floppy Disk Controller Data Sheet," Intel Corporation, 1980.
2. Intel, *iSBC 208 Hardware Reference Manual,* Manual Order No. 143078, Intel Corporation, 1980.
3. Intel, *iSBC 204 Flexible Diskette Controller Hardware Reference Manual,* Manual Order No. 9800568A, Intel Corporation, 1978.
4. Shugart, *SA800/801 Diskette Storage Drive OEM Manual,* Part No. 50574, Shugart Associates, 1977.
5. Shugart, *SA800/801 Diskette Storage Drive Theory of Operations,* Part No. 50664, Shugart Associates, 1977.
6. Shugart, *SA800 Series Diskette Storage Drive Double Density Design Guide,* Part No. 39000, Shugart Associates, 1977.
7. Shugart, "Application Notes for Shugart Dual VFO," Part No. 39101, Shugart Associates, 1980.
8. Pertec, "Soft-sector Formatting for PERTEC Flexible Disk Drives," Pertec Application Note, 1977.
9. Austin Lesea and Rodnay Zaks, "Floppy-disc Controller Design Must Begin With the Basics," EDN, May 20, 1978.
10. John Hoeppner and Larry Wall, "Encoding/Decoding Techniques Double Floppy Disc Capacity," Computer Design, Feb 1980.
11. John Zarrella, *System Architecture,* Mirocomputer Applications, 1980.

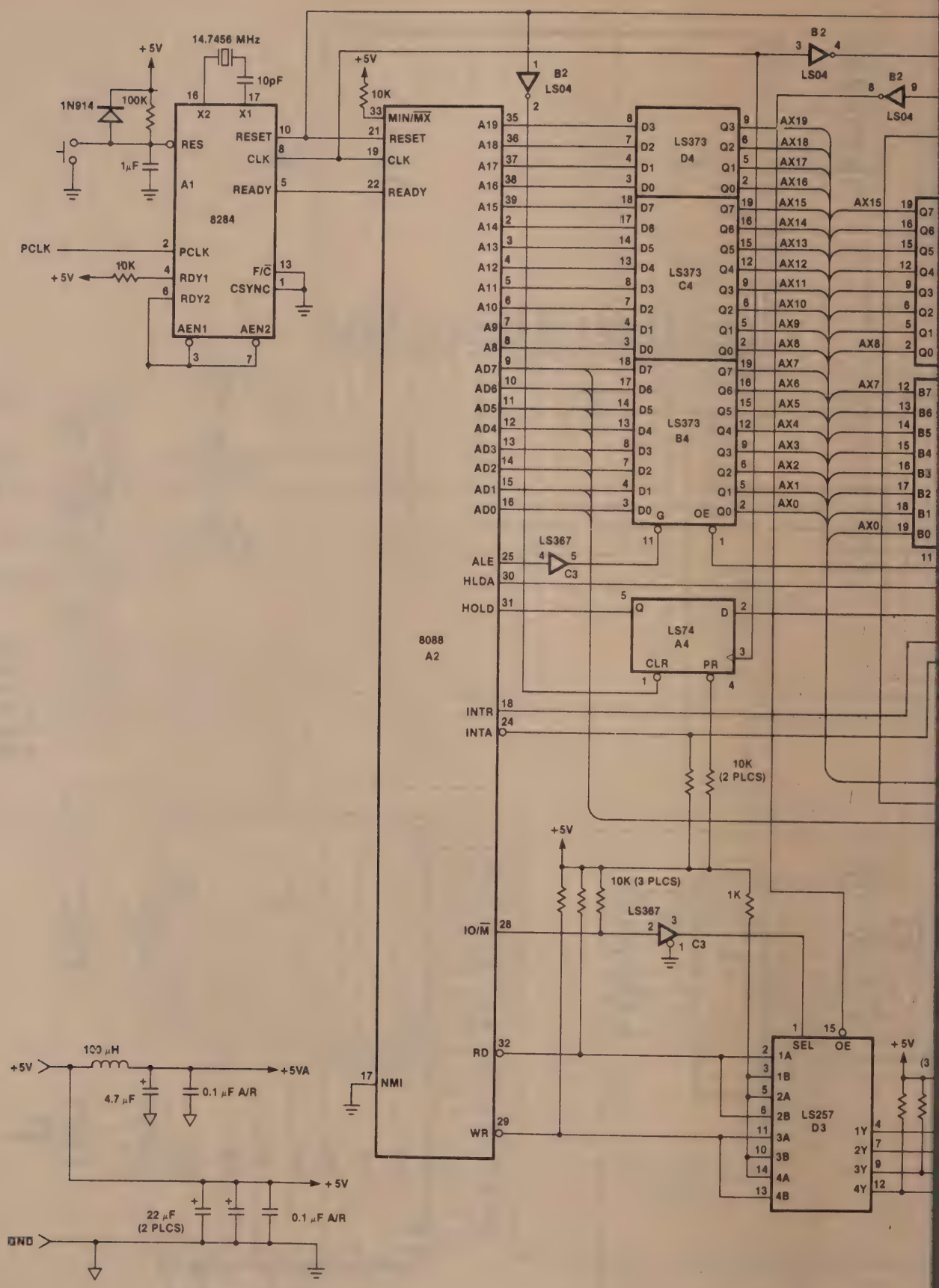

AD0-AD7
AD0
AD1
AD2
AD3
AD4
AD5
AD6
AD7
RESET
IOR
IOW
72CS
AX0
DREQ72
DACK72
INT72
EOP
D0
D1
D2
D3
D4
D5
D6
D7
RESET
RD
WR
CS
A0
DRQ
DACK
INT
TC
HDL
DS1
DS0
HDSEL
FR/STEP
LCT/DIR
RW/SEEK
WE
PS0
PS1
CLK
WR DATA
RDY
IDX
WP/TS
FLT/TRK0
WRCLK
DW
MFM
RD DATA
VCO
8272
D10
7438
H6
LS139
E10
LS04 (PLCS)
E6
H10
4.7K (4 PLCS)
+5V
1/2 LS240
G10
H8
H9
LS04 (4 PLCS)
E8
LS
E9
LS04
LS367
LS08 (2 PLCS)
E4
8224
X1
VDD
OSC
I6
RDYIN
RESIN
X2 SYNC
8 MHz
10 pF
4 MHz
LS393
Q0
I5
CLR
LS
175
G4
4D
3D
2D
4Q
3Q
2Q
150
(2 PLCS)
7414 (2 PLCS)
H7
READY
INDEX
150
(4 PLCS)
WRITE PROT
TWO SIDED
FAULT
TRACK0
1/2 LS240
G10
1A3
2A3
1A4
2A4
1Y3
2Y3
1Y4
2Y4
47 pF
3K
96LS02
G7
F7
LS393
Q0
Q1
1 MHz
500 kHz
1K
27K
200 pF
FREQ
DISC
PR
LS
74
G5
2F
1F
LS04
F8
150
7414
H7
READ DATA

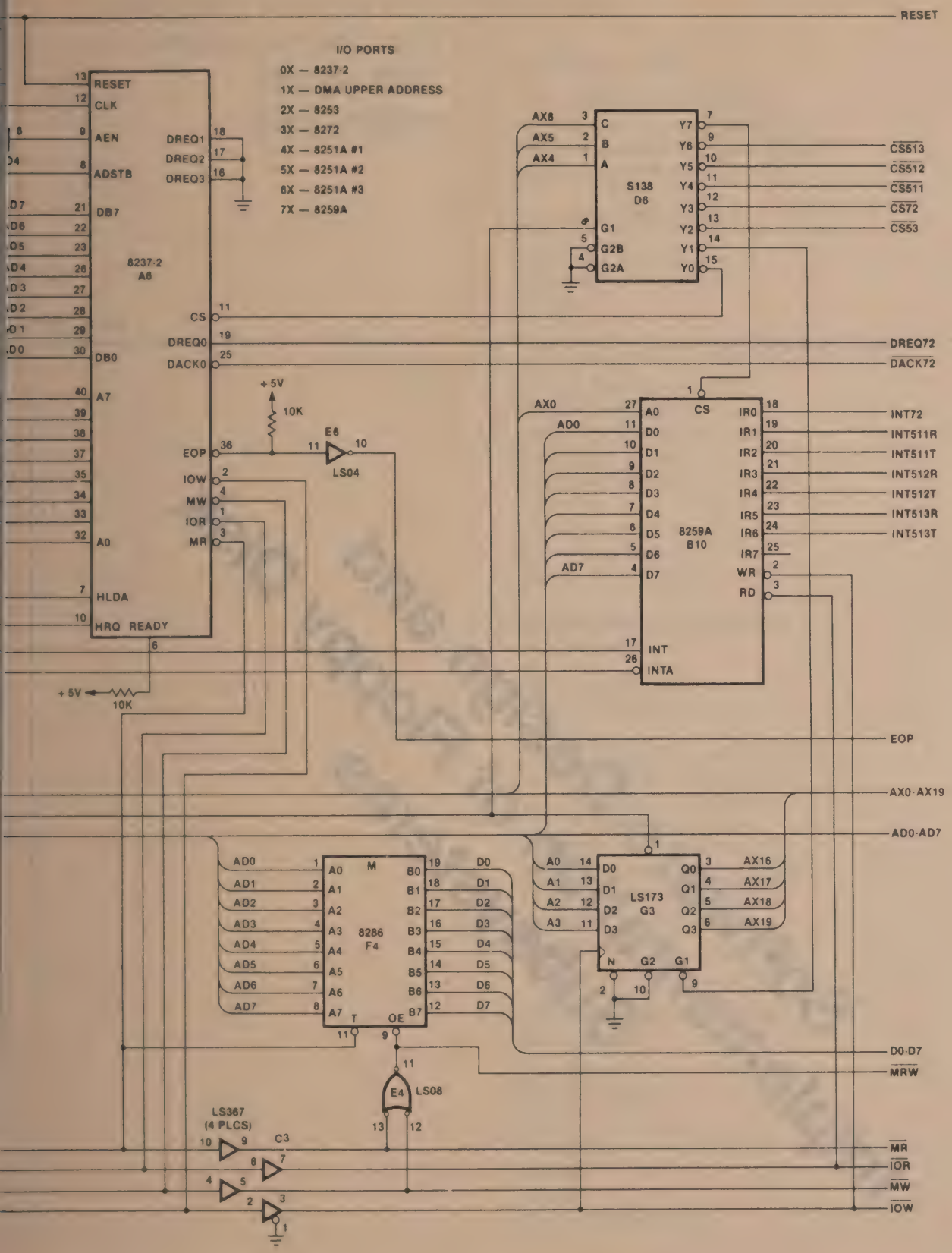

DSEL0
DSEL1
DSEL2
DSEL3
DIR
STEP
WRGT
FRES
LCT
SSEL
HDL
26 DRIVE SELECT0
28 DRIVE SELECT1
30 DRIVE SELECT2
32 DRIVE SELECT3
34 DIRECTION SELECT
36 STEP
40 WRITE GATE
4 FAULT RESET
2 LOW CURRENT
14 SIDE SELECT
18 HEAD LOAD
LS27 (3 PLCS)
LS04 (2 PLCS)
LS164
F5
E5
F8
E8
B1
LS32
LS157
3B
3A
3Y
F6
38 WRITE DATA
+5V
+5VA
1K
20K
1M
7438
H6
LS74
PR
CLR
PU
PUCLR
1N4148
(3 PLCS)
.01
.001
2K
10K
LM358
H5
LS04
E1
LS00
PD
PDCLR
DW
G5
27 pF
4.7K
5K
96LS02
G6
PLCLK
2 MHz NOM
CNTEN
CNT8
LS393
F7
A4
SETUP
ADJUSTMENT


AD0-AD7
AD0
AD1
AD2
AD3
AD4
AD5
AD6
AD7
D0
D7
8253
A10
BAUD RATE
GENERATOR
+5V
1K
GATE2
GATE1
GATE0
CLK2
CLK1
CLK0
A1
A0
OUT0
OUT1
OUT2
CS RD WR
AX1
AX0
CS511
CS512
CS513
AX0-AX19
PCLK
RESET
IOR
IOW
D0-D7
B1
LS232
(2 PLCS)
MRW
MW
MR
AX12
AX13
AX14
AX15
AX16
AX17
AX18
AX19
E1
LS00
(2 PLCS)
S138
E3
G1
G2B
G2A
Y0
Y7
F8
F9
FA
FB
FC
FD
FE
FF
PROM ADDRESS
DECODE
AX11
AX10
AX9
AX8
AX7
AX6
AX5
AX4
AX3
AX2
AX1
AX0
CE OE
A11
2732A
D1
2732A
D2
FE000
FF000
PROGRAM
MEMORY
D0
D1
D2
D3
D4
D5
D6
D7


AD0
AD1
AD2
AD3
AD4
AD5
AD6
AD7
D0
D7
RESET
RD
WR
A0
CLK
CTS
8251A
A9
8251A
B9
8251A
C9
SERIAL
PORTS
TXC
RXC
CS
TXRDY
RXRDY
RXD
TXD
1488
H3
H4
1489
330 PF
TXD3
RXD1
TXD2
RXD2
TXD1
RXD1
INT513R
INT513T
INT512R
INT512T
INT511R
INT511T
LS32
B1
E2
LS04
E6
LS27
G2A
G2B
G1
C
B
A
S138
F3
Y0
Y7
S0
S1
S2
S3
S4
S5
S6
S7
WE
A9
AX8
AX7
AX6
AX5
AX4
AX3
AX2
AX1
AX0
F1
G1
H1
I1
A0
F2
G2
H2
I2
2114 ARRAY
00000
00400
00800
00C00
01000
01400
01800
01C00
D4
D0
D7
D6
D5
D3
D2
D1
DATA
MEMORY

207875-002


14.7456 MHz
10pF
+5V
1N914
100K
1μF
X2
X1
RES
RESET
CLK
A1
READY
8284
PCLK
RDY1
RDY2
F/C̄
CSYNC
AEN1
AEN2
10K
MIN/MX̄
8088
A2
ALE
HLDA
HOLD
INTR
INTA
IO/M̄
RD
WR
NMI
LS373
D4
C4
B4
LS367
C3
LS74
A4
CLR
PR
B2
LS04
AX19
AX18
AX17
AX16
AX15
AX14
AX13
AX12
AX11
AX10
AX9
AX8
AX7
AX6
AX5
AX4
AX3
AX2
AX1
AX0
10K (2 PLCS)
10K (3 PLCS)
1K
LS257
D3
SEL
OE
100 μH
4.7 μF
0.1 μF A/R
+5VA
22 μF
(2 PLCS)
GND


RESET
I/O PORTS
0X — 8237-2
1X — DMA UPPER ADDRESS
2X — 8253
3X — 8272
4X — 8251A #1
5X — 8251A #2
6X — 8251A #3
7X — 8259A
8237-2
A6
RESET
CLK
AEN
ADSTB
DB7
DB0
A7
A0
HLDA
HRQ READY
DREQ1
DREQ2
DREQ3
CS
DREQ0
DACK0
EOP
IOW
MW
IOR
MR
S138
D6
AX8
AX5
AX4
CS513
CS512
CS511
CS72
CS53
DREQ72
DACK72
+5V
10K
E6
LS04
8259A
B10
AX0
AD0
AD7
INT72
INT511R
INT511T
INT512R
INT512T
INT513R
INT513T
INT
INTA
EOP
AX0-AX19
AD0-AD7
8286
F4
LS173
G3
AX16
AX17
AX18
AX19
D0-D7
MRW
E4
LS08
LS367
(4 PLCS)
C3
MR
IOR
MW
IOW

intel

APPLICATION NOTE

AP-121

June 1981

Software Design and Implementation of Floppy Disk Subsystems

© Intel Corporation, 1981.

207885-001

Software Design and Implementation of Floppy Disk Subsystems

Contents

1. INTRODUCTION

The Physical Interface Level
The Logical Interface Level
The File System Interface Level
Scope of this Note

2. DISK I/O TECHNIQUES

FDC Data Transfer Interface
Overlapped Operations
Buffers

3. THE 8272 FLOPPY DISK CONTROLLER

Floppy Disk Commands
Interface Registers
Command/Result Phases
Execution Phase
Multi-sector and Multi-track Transfers
Drive Status Polling
Command Details
Invalid Commands

4. 8272 PHYSICAL INTERFACE SOFTWARE

INITIALIZE$DRIVERS
EXECUTE$DOCB
FDCINT
OUTPUT$CONTROLS$TO$DMA
OUTPUT$COMMAND$TO$FDC
INPUT$RESULT$FROM$FDC
OUTPUT$BYTE$TO$FDC
INPUT$BYTE$FROM$FDC
FDC$READY$FOR$COMMAND
FDC$READY$FOR$RESULT
OPERATION$CLEAN$UP
Modifications for Polling Operation

5. 8272 LOGICAL INTERFACE SOFTWARE

SPECIFY
RECALIBRATE
SEEK
FORMAT
WRITE
READ
Coping With Errors

Contents (Continued)

6. FILE SYSTEMS

File Allocation
The Intel File System
Disk File System Functions

7. KEY 8272 SOFTWARE INTERFACING CONSIDERATIONS

REFERENCES

APPENDIX A—8272 FDC DEVICE DRIVER SOFTWARE

APPENDIX B—8272 FDC EXERCISER PROGRAM

APPENDIX C—8272 DRIVER FLOWCHARTS

1. Introduction

Disk interface software is a major contributor to the efficient and reliable operation of a floppy disk subsystem. This software must be a well-designed compromise between the needs of the application software modules and the capabilities of the floppy disk controller (FDC). In an effort to meet these requirements, the implementation of disk interface software is often divided into several levels of abstraction. The purpose of this application note is to define these software interface levels and describe the design and implementation of a modular and flexible software driver for the 8272 FDC. This note is a companion to AP-116, "An Intelligent Data Base System Using the 8272."

The Physical Interface Level

The software interface level closest to the FDC hardware is referred to as the physical interface level. At this level, interface modules (often called disk drivers or disk handlers) communicate directly with the FDC device. Disk drivers accept floppy disk commands from other software modules, control and monitor the FDC execution of the commands, and finally return operational status information (at command termination) to the requesting modules.

In order to perform these functions, the drivers must support the bit/byte level FDC interface for status and data transfers. In addition, the drivers must field, classify, and service a variety of FDC interrupts.

The Logical Interface Level

System and application software modules often specify disk operation parameters that are not directly compatible with the FDC device. This software incompatibility is typically caused by one of the following:

1. The change from an existing FDC to a functionally equivalent design. Replacing a TTL based controller with an LSI device is an example of a change that may result in software incompatibilities.

2. The upgrade of an existing FDC subsystem to a higher capability design. An expansion from a single-sided, single-density system to a dual-sided, double-density system to increase data storage capacity is an example of such a system change.

3. The abstraction of the disk software interface to avoid redundancy. Many FDC parameters (in particular the density, gap size, number of sectors per track and number of bytes per sector) are fixed for a floppy disk (after formatting). In fact, in many systems these parameters are never changed during the life of the system.

4. The requirement to support a software interface that is independent of the type of disk attached to the system. In this case, a system generated ("logical") disk address (drive, head, cylinder, and sector numbers) must be mapped into a physical floppy disk address. For example, to switch between single- and dual-sided disks, it may be easier and more cost-effective for the software to treat the dual-sided disk as containing twice as many sectors per track (52) rather than as having two sides. With this technique, accesses to sectors 1 through 26 are mapped onto head 0 while accesses to sectors 27 through 52 are mapped onto head 1.

5. The necessity of supporting a bad track map. Since bad tracks depend on the disk media, the bad track mapping varies from disk to disk. In general, the system and application software should not be concerned with calculating bad track parameters. Instead, these software modules should refer to cylinders logically (0 through 76). The logical interface level procedures must map these cylinders into physical cylinder positions in order to avoid the bad tracks.

The key to logical interface software design is the mapping of the "logical disk interface" (as seen by the application software) into the "physical disk interface" (as implemented by the floppy disk drivers). This logical to physical mapping is tightly coupled to system software design and the mapping serves to isolate both applications and system software from the peculiarities of the FDC device. Typical logical interface procedures are described in Table 1.

The File System Interface Level

The file system typically comprises the highest level of disk interface software used by application programs. The file system is designed to treat the disk as a collection of named data areas (known as files). These files are cataloged in the disk directory. File system interface software permits the creation of new files and the deletion of existing files under software control. When a file is created, its name and disk address are entered into the directory; when a file is deleted, its name is removed from the directory. Application software requests the use of a file by executing an OPEN function. Once opened, a file is normally reserved for use by the requesting program or task and the file cannot be reopened by other tasks. When a task no longer needs to use an open file, the task closes the file, releasing it for use by other tasks.

Most file systems also support a set of file attributes that can be specified for each file. File attributes may be used to protect files (e.g., the WRITE PROTECT attribute ensures that an existing file cannot accidentally be overwritten) and to supply system configuration information (e.g., a FORMAT attribute may specify that a file should automatically be created on a new disk when the disk is formatted).

At the file system interface level, application programs need not be explicitly aware of disk storage allocation techniques, block sizes, or file coding strategies. Only a "file name" must be presented in order to open, read or write, and subsequently close a file. Typical file system functions are listed in Table 2.

Table 1: Examples of Logical Interface Procedures

Name	Description
FORMAT DISK	Controls physical disk formatting for all tracks on a disk. Formatting adds FDC recognized cylinder, head, and sector addresses as well as address marks and data synchronization fields (gaps) to the floppy disk media.
RECALIBRATE	Moves the disk read/write head to track 0 (at the outside edge of the disk).
SEEK	Moves the disk read/write head to a specified logical cylinder. The logical and physical cylinder numbers may be different if bad track mapping is used.
READ STATUS	Indicates the status of the floppy disk drive and media. One important use of this procedure is to determine whether a floppy disk is dual-sided.
READ SECTOR	Reads one or more complete sectors starting at a specified disk address (drive, head, cylinder, and sector).
WRITE SECTOR	Writes one or more complete sectors starting at a specified disk address (drive, head, cylinder, and sector).

Table 2: Disk File System Functions

Name	Description
OPEN	Prepare a file for processing. If the file is to be opened for input and the file name is not found in the directory, an error is generated. If the file is opened for output and the file name is not found in the directory, the file is automatically created.
CLOSE	Terminate processing of an open file.
READ	Transfer data from an open file to memory. The READ function is often designed to buffer one or more sectors of data from the disk drive and supply this data to the requesting program, as required.
WRITE	Transfer data from memory to an open file. The WRITE function is often designed to buffer data from the application program until enough data is available to fill a disk sector.
CREATE	Initialize a file and enter its name and attributes into the file directory.
DELETE	Remove a file from the directory and release its storage space.
RENAME	Change the name of a file in the directory.
ATTRIBUTE	Change the attributes of a file.
LOAD	Read a file of executable code into memory.
INITDISK	Initialize a disk by formatting the media and establishing the directory file, the bit map file, and other system files.

Scope of this Note

This application note directly addresses the logical and physical interface levels. A complete 8272 driver (including interrupt service software) is listed in Appendix A. In addition, examples of recalibrate, seek, format, read, and write logical interface level procedures are included as part of the exerciser program found in Appendix B. Wherever possible, specific hardware configuration dependencies are parametized to provide maximum flexibility without requiring major software changes.

2. Disk I/O Techniques

One of the most important software aspects of disk interfacing is the fixed sector size. (Sector sizes are fixed when the disk is formatted.) Individual bytes of disk storage cannot be read/written; instead, complete sectors must be transferred between the floppy disk and system memory.

Selection of the appropriate sector size involves a tradeoff between memory size, disk storage efficiency, and disk transfer efficiency. Basically, the following factors must be weighed:

1. Memory size. The larger the sector size, the larger the memory area that must be reserved for use during disk I/O transfers. For example, a 1K byte disk sector size requires that at least one 1K memory block be reserved for disk I/O.

2. Disk Storage efficiency. Both very large and very small sectors can waste disk storage space as follows. In disk file systems, space must be allocated somewhere on the disk to link the sectors of each file together. If most files are composed of many small sectors, a large amount of linkage overhead information is required. At the other extreme, when most files are smaller than a single disk sector, a large amount of space is wasted at the end of each sector.

3. Disk transfer efficiency. A file composed of a few large sectors can be transferred to/from memory more efficiently (faster and with less overhead) than a file composed of many small sectors.

Balancing these considerations requires knowledge of the intended system applications. Typically, for general purpose systems, sector sizes from 128 bytes to 1K bytes are used. For compatibility between single-density and double-density recording with the 8272 floppy disk controller, 256 byte sectors or 512 byte sectors are most useful.

FDC Data Transfer Interface

Three distinct software interface techniques may be used to interface system memory to the FDC device during sector data transfers:

1. DMA - In a DMA implementation, the software is only required to set up the DMA controller memory address and transfer count, and to initiate the data transfer. The DMA controller hardware handshakes with the processor/system bus in order to perform each data transfer.

2. Interrupt Driven - The FDC generates an interrupt when a data byte is ready to be transferred to memory, or when a data byte is needed from memory. It is the software's responsibility to perform appropriate memory reads/writes in order to transfer data from/to the FDC upon receipt of the interrupt.

3. Polling - Software responsibilities in the polling mode are identical to the responsibilities in the interrupt driven mode. The polling mode, however, is used when interrupt service overhead (context switching) is too large to support the disk data

rate. In this mode, the software determines when to transfer data by continually polling a data request status flag in the FDC status register.

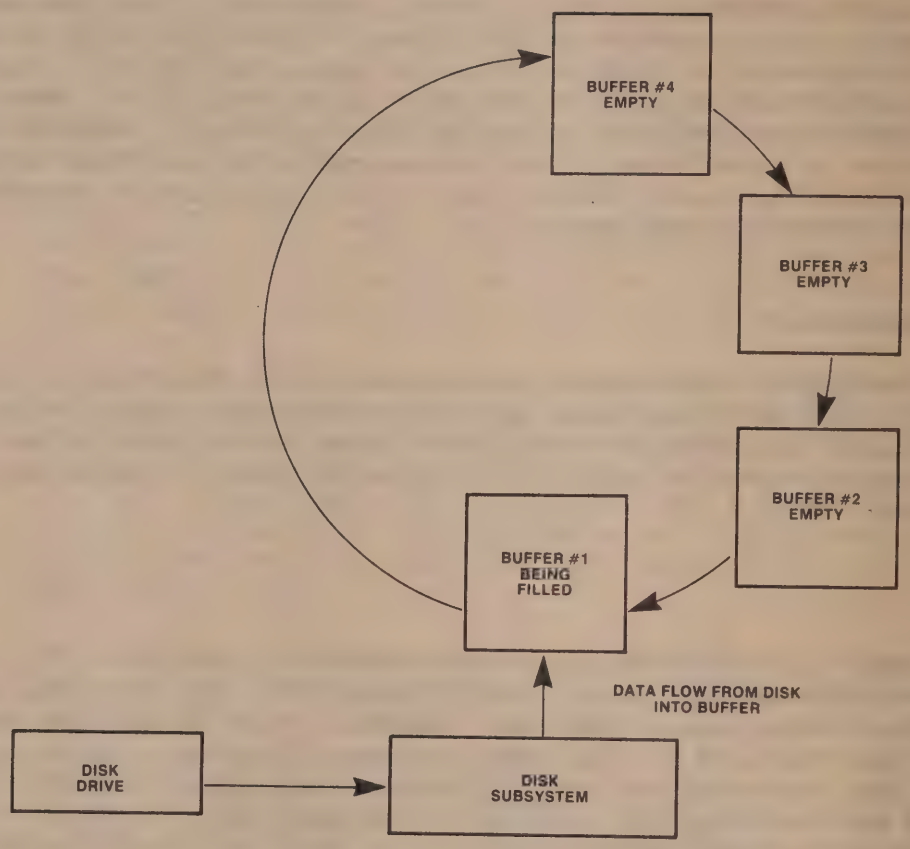
The DMA mode has the advantage of permitting the processor to continue executing instructions while a disk transfer is in progress. (This capability is especially useful in multiprogramming environments when the operating system is designed to permit other tasks to execute while a program is waiting for I/O.) Modes 2 and 3 are often combined and described as non-DMA operating modes. Non-DMA modes have the advantage of significantly lower system cost, but are often performance limited for double-density systems (where data bytes must be transferred to/from the FDC every 16 microseconds).

Overlapped Operations

Some FDC devices support simultaneous disk operations on more than one disk drive. Normally seek and recalibrate operations can be overlapped in this manner. Since seek operations on most floppy drives are extremely slow, this mode of operation can often be used by the system software to reduce overall disk access times.

Buffers

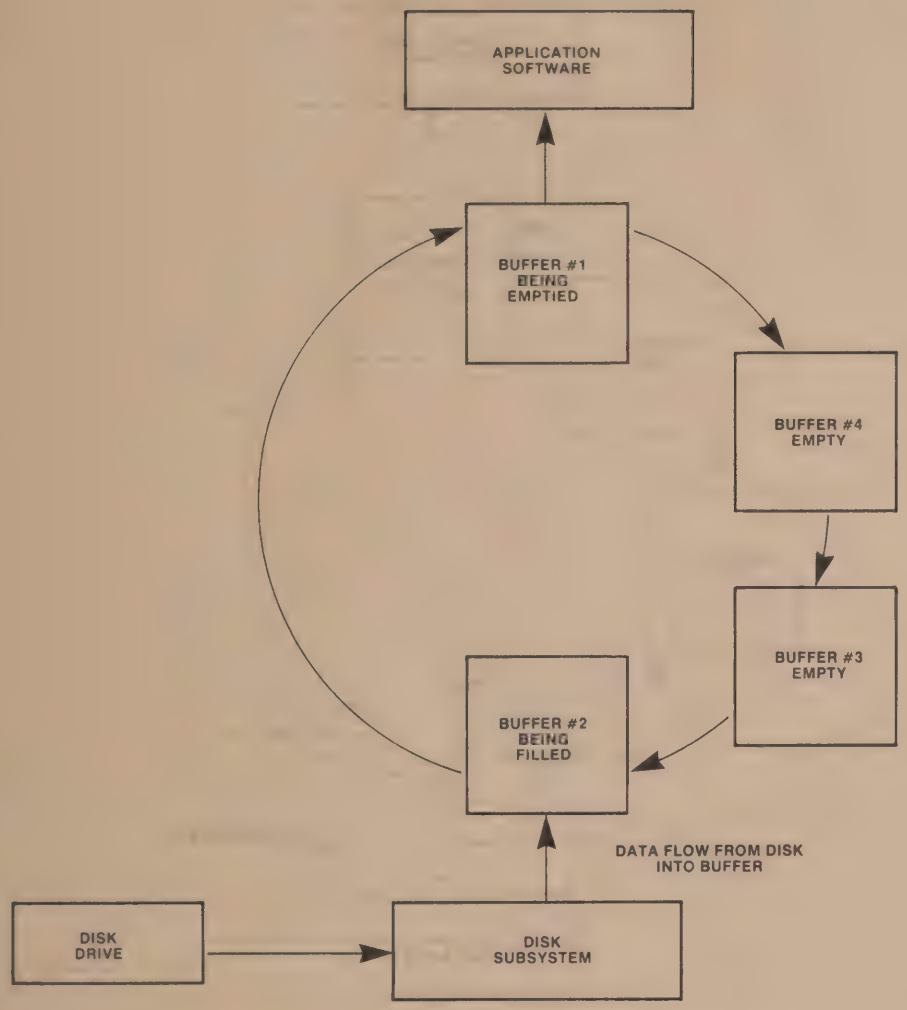
The buffer concept is an extremely important element in advanced disk I/O strategies. A buffer is nothing more than a memory area containing the same amount of data as a disk sector contains. Generally, when an application program requests data from a disk, the system software allocates a buffer (memory area) and transfers the data from the appropriate disk sector into the buffer. The address of the buffer is then returned to the application software. In the same manner, after the application program has filled a buffer for output, the buffer address is passed to the system software, which writes data from the buffer into a disk sector. In multitasking systems, multiple buffers may be allocated from a buffer pool. In these systems, the disk controller is often requested to read ahead and fill additional data buffers while the application software is processing a previous buffer. Using this technique, system software attempts to fill buffers before they are needed by the application programs, thereby eliminating program waits during I/O transfers. Figure 1 illustrates the use of multiple buffers in a ring configuration.



a) The first disk read request by the application software causes the disk subsystem to begin filling the first empty buffer. The application software must wait until the buffer is filled before it may continue execution.

AFN-01949A

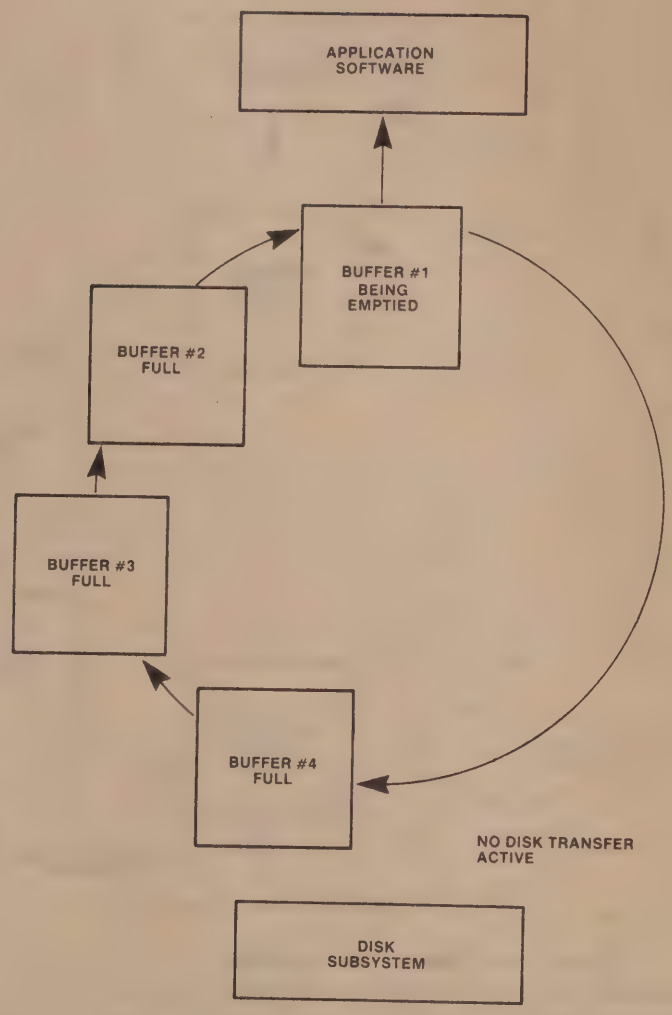
Figure 1. Using Multiple Memory Buffers for Disk I/O



b) After the first buffer is filled, the disk system continues to transfer disk data into the next buffer while the application software begins operating on the first full buffer.

AFN-01949A

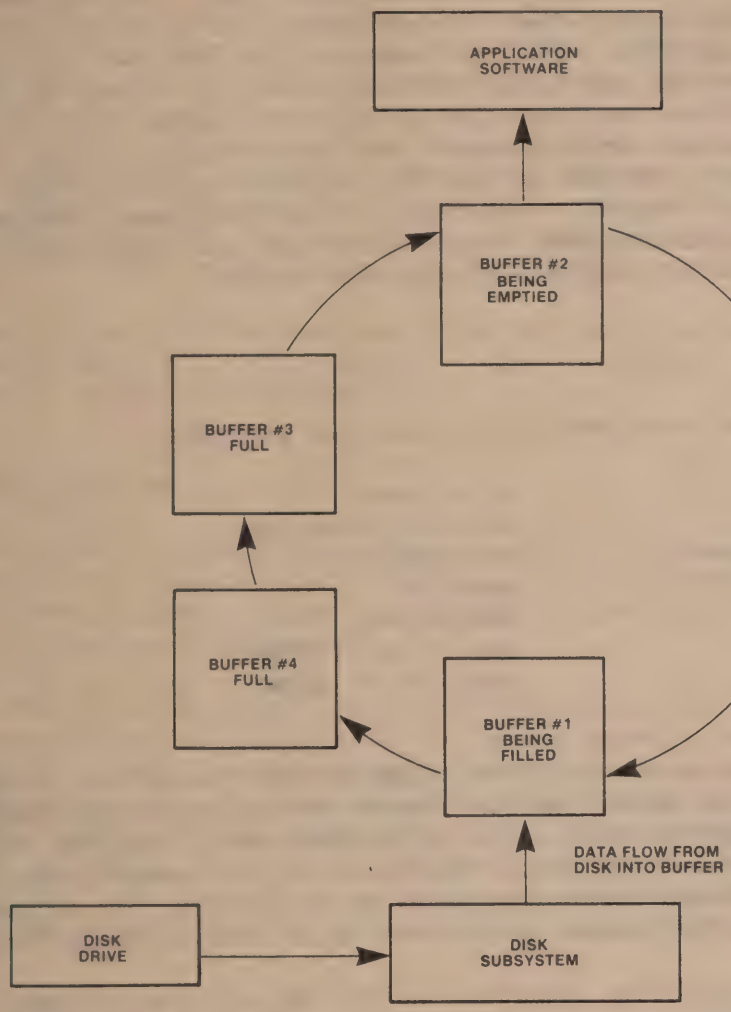
Figure 1. Using Multiple Memory Buffers for Disk I/O (Continued)



c) When all empty buffers have been filled, disk activity is stopped until the application software releases one or more buffers for reuse.

AFN-01949A

Figure 1. Using Multiple Memory Buffers for Disk I/O (Continued)



d) When the application software releases a buffer (for reuse), the disk subsystem begins a disk sector read to refill the buffer. This strategy attempts to anticipate application software needs by maintaining a sufficient number of full data buffers in order to minimize data transfer delays. If disk data is already in memory when the application software requests it, no disk transfer delays are incurred.

AFN-01949A

Figure 1. Using Multiple Memory Buffers for Disk I/O (Continued)

3. THE 8272 FLOPPY DISK CONTROLLER

The 8272 is a single-chip LSI Floppy Disk Controller (FDC) that implements both single- and double-density floppy disk storage subsystems (with up to four dual-sided disk drives per FDC). The 8272 supports the IBM 3740 single-density recording format (FM) and the IBM System 34 double-density recording format (MFM). The 8272 accepts and executes high-level disk commands such as format track, seek, read sector, and write sector. All data synchronization and error checking is automatically performed by the FDC to ensure reliable data storage and subsequent retrieval. The 8272 interfaces to microprocessor systems with or without Direct Memory Access (DMA) capabilities and also interfaces to a large number of commercially available floppy disk drives.

Floppy Disk Commands

The 8272 executes fifteen high-level disk interface commands:

Specify	Write Data
Sense Drive Status	Write Deleted Data
Sense Interrupt Status	Read Track
Seek	Read ID
Recalibrate	Scan Equal
Format Track	Scan High or Equal
Read Data	Scan Low or Equal
Read Deleted Data	

Each command is initiated by a multi-byte transfer from the driver software to the FDC (the transferred bytes contain command and parameter information). After complete command specification, the FDC automatically executes the command. The command result data (after execution of the command) may require a multi-byte transfer of status information back to the driver. It is convenient to consider each FDC command as consisting of the following three phases:

Command Phase: The driver transfers to the FDC all the information required to perform a particular disk operation. The 8272 automatically enters the command phase after RESET and following the completion of the result phase (if any) of a previous command.

Execution Phase: The FDC performs the operation as instructed. The execution phase is entered immediately after the last command parameter is written to the FDC in the preceding command phase. The execution phase normally ends when the last data byte is transferred to/from the disk or when an error occurs.

Result Phase: After completion of the disk operation, status and other housekeeping information are made available to the driver software. After this information is read, the FDC reenters the command phase and is ready to accept another command.

Interface Registers

To support information transfer between the FDC and the system software, the 8272 contains two 8-bit registers: the Main Status Register and the Data Register. The Main Status Register (read only) contains FDC status information and may be accessed at any time. The Main Status Register (Table 3) provides the system processor with the status of each disk drive, the status of the FDC, and the status of the processor interface. The Data Register (read/write) stores data, commands, parameters, and disk drive status information. The Data Register is used to program the FDC during the command phase and to obtain result information after completion of FDC operations.

In addition to the Main Status Register, the FDC contains four additional status registers (ST0, ST1, ST2, and ST3). These registers are only available during the result phase of a command.

Command/Result Phases

Table 4 lists the 8272 command set. For each of the fifteen commands, command and result phase data transfers are listed. A list of abbreviations used in the table is given in Table 5, and the contents of the result status registers (ST0-ST3) are illustrated in Table 6.

The bytes of data which are sent to the 8272 by the drivers during the command phase, and are read out of the 8272 in the result phase, must occur in the order shown in Table 4. That is, the command code must be sent first and the other bytes sent in the prescribed sequence. All bytes of the command and result phases must be read/written as described. After the last byte of data in the command phase is sent to the 8272 the execution phase automatically starts. In a similar fashion, when the last byte of data is read from the 8272 in the result phase, the result phase is automatically ended and the 8272 reenters the command phase.

It is important to note that during the result phase all bytes shown in Table 4 must be read. The Read Data command, for example, has seven bytes of data in the result phase. All seven bytes must be read in order to successfully complete the Read Data command. The 8272 will not accept a new command until all seven bytes have been read. The number of command and result bytes varies from command-to-command.

In order to read data from, or write data to, the Data Register during the command and result phases, the software driver must examine the Main Status Register to determine if the Data Register is available. The DIO (bit 6) and RQM (bit 7) flags in the Main Status Register must be low and high, respectively, before each byte of the command word may be written into the 8272. Many of the commands require multiple bytes, and as a result, the Main Status Register must be read prior to each byte transfer to the 8272. To read status bytes during the result phase, DIO and RQM in the Main Status Register must both be high. Note, checking the Main Status Register in this manner before each byte transfer to/from the 8272 is required only in the command and result phases, and is NOT required during the execution phase.

Table 3: Main Status Register Bit Definitions

BIT NUMBER	SYMBOL	DESCRIPTION
0	D_0B	Disk Drive 0 Busy. Disk Drive 0 is seeking.
1	D_1B	Disk Drive 1 Busy. Disk Drive 1 is seeking.
2	D_2B	Disk Drive 2 Busy. Disk Drive 2 is seeking.
3	D_3B	Disk Drive 3 Busy. Disk Drive 3 is seeking.
4	CB	FDC Busy. A read or write command is in progress.
5	NDM	Non-DMA Mode. The FDC is in the non-DMA mode when this flag is set (1). This flag is set only during the execution phase of commands in the non-DMA mode. Transition of this flag to a zero (0) indicates that the execution phase has ended.
6	DIO	Data Input/Output. Indicates the direction of a data transfer between the FDC and the Data Register. When DIO is set (1), data is read from the Data Register by the processor; when DIO is reset (0), data is written from the processor to the Data Register.
7	RQM	Request for Master. When set (1), this flag indicates that the Data Register is ready to send data to, or receive data from, the processor.

Table 4: 8272 Command Set

PHASE	R/W	DATA BUS D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	REMARKS
READ DATA										
Command	W	MT	MFM	SK	0	0	1	1	0	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W	C								Sector ID information prior to Command execution
	W	H								
	W	R								
	W	N								
	W	EOT								
	W	GPL								
	W	DTL								
Execution										Data transfer between the FDD and the main-system
Result	R	ST 0								Status information after Command execution
	R	ST 1								
	R	ST 2								
	R	C								Sector ID information after command execution
	R	H								
	R	R								
	R	N								
READ DELETED DATA										
Command	W	MT	MFM	SK	0	1	1	0	0	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W	C								Sector ID information prior to Command execution
	W	H								
	W	R								
	W	N								
	W	EC 1								
	W	GPL								
	W	DTL								
Execution										Data transfer between the FDD and the main-system
Result	R	ST 0								Status information after Command execution
	R	ST 1								
	R	ST 2								
	R	C								Sector ID information after Command execution
	R	H								
	R	R								
	R	N								
WRITE DATA										
Command	W	MT	MFM	0	0	0	1	0	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W	C								Sector ID information prior to Command execution
	W	H								
	W	R								
	W	N								
	W	EOT								
	W	GPL								
	W	DTL								
Execution										Data transfer between the main-system and the FDD
Result	R	ST 0								Status information after Command execution
	R	ST 1								
	R	ST 2								
	R	C								Sector ID information after Command execution
	R	H								
	R	R								
	R	N								
WRITE DELETED DATA										
Command	W	MT	MFM	0	0	1	0	0	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W	C								Sector ID information prior to Command execution
	W	H								
	W	R								
	W	N								
	W	EOT								
	W	GPL								
	W	DTL								
Execution										Data transfer between the FDD and the main-system
Result	R	ST 0								Status information after Command execution
	R	ST 1								
	R	ST 2								
	R	C								Sector ID information after Command execution
	R	H								
	R	R								
	R	N								

PHASE	R/W	DATA BUS D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	REMARKS
READ A TRACK										
Command	W	0	MFM	SK	0	0	0	1	0	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W	C								Sector ID information prior to Command execution
	W	H								
	W	R								
	W	N								
	W	EOT								
	W	GPL								
	W	DTL								
Execution										Data transfer between the FDD and the main-system. FDC reads the complete track contents from the physical index mark to EOT
Result	R	ST 0								Status information after Command execution
	R	ST 1								
	R	ST 2								
	R	C								Sector ID information after Command execution
	R	H								
	R	R								
	R	N								
READ ID										
Command	W	0	MFM	0	0	1	0	1	0	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
Execution										The first correct ID information on the track is stored in Data Register
Result	R	ST 0								Status information after Command execution
	R	ST 1								
	R	ST 2								
	R	C								Sector ID information during Execution Phase
	R	H								
	R	R								
	R	N								
FORMAT A TRACK										
Command	W	0	MFM	0	0	1	1	0	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W	N								Bytes/Sector
	W	SC								Sectors/Track
	W	GPL								Gap 3
	W	D								Filter Byte
Execution										FDC formats an entire track
Result	R	ST 0								Status information after Command execution
	R	ST 1								
	R	ST 2								
	R	C								In this case, the ID information has no meaning
	R	H								
	R	R								
	R	N								
SCAN EQUAL										
Command	W	MT	MFM	SK	1	0	0	0	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W	C								Sector ID information prior to Command execution
	W	H								
	W	R								
	W	N								
	W	EOT								
	W	GPL								
	W	STP								
Execution										Data compared between the FDD and the main-system
Result	R	ST 0								Status information after Command execution
	R	ST 1								
	R	ST 2								
	R	C								Sector ID information after Command execution
	R	H								
	R	R								
	R	N								

Note: 1. $A_0 = 1$ for all operations.

PHASE	R/W	DATA BUS D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	REMARKS
SCAN LOW OR EQUAL										
Command	W	MT	MFM	SK	1	1	0	0	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W	C								Sector ID information prior Command execution
	W	H								
	W	R								
	W	N								
	W	EOT								
	W	GPL								
	W	STP								
Execution										Data compared between the FDD and the main-system
Result	R	ST 0								Status information after Command execution
	R	ST 1								
	R	ST 2								
	R	C								Sector ID information after Command execution
	R	H								
	R	R								
	R	N								
SCAN HIGH OR EQUAL										
Command	W	MT	MFM	SK	1	1	1	0	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W	C								Sector ID information prior Command execution
	W	H								
	W	R								
	W	N								
	W	EOT								
	W	GPL								
	W	STP								
Execution										Data compared between the FDD and the main-system
Result	R	ST 0								Status information after Command execution
	R	ST 1								
	R	ST 2								
	R	C								Sector ID information after Command execution
	R	H								
	R	R								
	R	N								

PHASE	R/W	DATA BUS D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	REMARKS
RECALIBRATE										
Command	W	0	0	0	0	0	1	1	1	Command Codes
	W	0	0	0	0	0	0	DS1	DS0	
Execution										Head retracted to Track 0
SENSE INTERRUPT STATUS										
Command	W	0	0	0	0	1	0	0	0	Command Codes
Result	R	ST 0								Status information at the end of each seek operation about the FDC
	R	C								
SPECIFY										
Command	W	0	0	0	0	0	0	1	1	Command Codes
	W	SPT						HUT		Timer Settings
	W	HLT							ND	
SENSE DRIVE STATUS										
Command	W	0	0	0	0	0	1	0	0	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
Result	R	ST 3								Status information about the FDD
SEEK										
Command	W	0	0	0	0	1	1	1	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W	C								
Execution										Head is positioned over proper Cylinder on Diskette
INVALID										
Command	W	Invalid Codes								Invalid Command Codes (NoOp — FDC goes into Standby State)
Result	R	ST 0								ST 0 = 80 (16)

Table 5: Command/Result Parameter Abbreviations

SYMBOL	DESCRIPTION
C	Cylinder Address. The currently selected cylinder address (0 to 76) on the disk.
D	Data Pattern. The pattern to be written in each sector data field during formatting.
DS0,DS1	Disk Drive Select. DS1 DS0 0 0 Drive 0 0 1 Drive 1 1 0 Drive 2 1 1 Drive 3
DTL	Special Sector Size. During the execution of disk read/write commands, this parameter is used to temporarily alter the effective disk sector size. By setting N to zero, DTL may be used to specify a sector size from 1 to 256 bytes in length. If the actual sector (on the disk) is larger than DTL specifies, the remainder of the actual sector is not passed to the system during read commands; during write commands, the remainder of the actual sector is written with all-zeroes bytes. DTL should be set to FF hexadecimal when N is not zero.
EOT	End of Track. The final sector number of the current track.
GPL	Gap Length. The gap 3 size. (Gap 3 is the space between sectors.)
H	Head Address. Selected head: 0 or 1 (disk side 0 or 1, respectively) as encoded in the sector ID field.
HLT	Head Load Time. Defines the time interval that the FDC waits after loading the head before initiating a read or write operation. Programmable from 2 to 254 milliseconds (in increments of 2 ms).
HUT	Head Unload Time. Defines the time interval from the end of the execution phase (of a read or write command) until the head is unloaded. Programmable from 16 to 240 milliseconds (in increments of 16 ms).
MFM	MFM/FM Mode Selector. Selects MFM double-density recording mode when high, FM single-density mode when low.
MT	Multi-Track Selector. When set, this flag selects the multi-track operating mode. In this mode (used only with dual-sided disks), the FDC treats a complete cylinder (under both read/write head 0 and read/write head 1) as a single track. The FDC operates as if this expanded track started at the first sector under head 0 and ended at the last sector under head 1. With this flag set (high), a multi-sector read operation will automatically continue to the first sector under head 1 when the FDC finishes operating on the last sector under head 0.
N	Sector Size Code. The number of data bytes within a sector.

ND	Non-DMA Mode Flag. When set (1), this flag indicates that the FDC is to operate in the non-DMA mode. In this mode, the processor participates in each data transfer (by means of an interrupt or by polling the RQM flag in the Main Status Register). When reset (0), the FDC interfaces to a DMA controller.
R	Sector Address. Specifies the sector number to be read or written. In multi-sector transfers, this parameter specifies the sector number of the first sector to be read or written.
SC	Number of Sectors per Track. Specifies the number of sectors per track to be initialized by the Format Track command.
SK	Skip Flag. When this flag is set, sectors containing deleted data address marks will automatically be skipped during the execution of multi-sector Read Data or Scan commands. In the same manner, a sector containing a data address mark will automatically be skipped during the execution of a multi-sector Read Deleted Data command.
SRT	Step Rate Interval. Defines the time interval between step pulses issued by the FDC (track-to-track access time). Programmable from 1 to 16 milliseconds (in increments of 1 ms).
ST0 ST1 ST2 ST3	Status Register 0-3. Registers within the FDC that store status information after a command has been executed. This status information is available to the processor during the Result Phase after command execution. These registers may only be read after a command has been executed (in the exact order shown in Table 4 for each command). These registers should not be confused with the Main Status Register.
STP	Scan Sector Increment. During Scan operations, this parameter is added to the current sector number in order to determine the next sector to be scanned.

Table 6: Status Register Definitions

Status Register 0		
BIT NUMBER	SYMBOL	DESCRIPTION
7,6	IC	Interrupt Code. 00 - Normal termination of command. The specified command was properly executed and completed without error. 01 - Abnormal termination of command. Command execution was started but could not be successfully completed. 10 - Invalid command. The requested command could not be executed. 11 - Abnormal termination. During command execution, the disk drive ready signal changed state.
5	SE	Seek End. This flag is set (1) when the FDC has completed the Seek command and the read/write head is positioned over the correct cylinder.
4	EC	Equipment Check Error. This flag is set (1) if a fault signal is received from the disk drive or if the track 0 signal is not received from the disk drive after 77 step pulses (Recalibrate command).
3	NR	Not Ready Error. This flag is set if a read or write command is issued and either the drive is not ready or the command specifies side 1 (head 1) of a single-sided disk.
2	H	Head Address. The head address at the time of the interrupt.
1,0	DS1,DS0	Drive Select. The number of the drive selected at the time of the interrupt.

Status Register 1		
BIT NUMBER	SYMBOL	DESCRIPTION
7	EN	End of Track Error. This flag is set if the FDC attempts to access a sector beyond the final sector of the track.
6		Undefined
5	DE	Data Error. Set when the FDC detects a CRC error in either the the ID field or the data field of a sector.
4	OR	Overrun Error. Set (during data transfers) if the FDC does not receive DMA or processor service within the specified time interval.

3		Undefined
2	ND	Sector Not Found Error. This flag is set by any of the following conditions. a) The FDC cannot locate the sector specified in the Read Data, Read Deleted Data, or Scan command. b) The FDC cannot locate the starting sector specified in the Read Track command. c) The FDC cannot read the ID field without error during a Read ID command.
1	NW	Write Protect Error. This flag is set if the FDC detects a write protect signal from the disk drive during the execution of a Write Data, Write Deleted Data, or Format Track command.
0	MA	Missing Address Mark Error. This flag is set by either of the following conditions: a) The FDC cannot detect the ID address mark on the specified track (after two rotations of the disk). b) The FDC cannot detect the data address mark or deleted data address mark on the specified track. (See also the MD bit of Status Register 2.)

Status Register 2

BIT NUMBER	SYMBOL	DESCRIPTION
7		Undefined
6	CM	Control Mark. This flag is set when the FDC encounters one of the following conditions: a) A deleted data address mark during the execution of a Read Data or Scan command. b) A data address mark during the execution of a Read Deleted Data command.
5	DD	Data Error. Set (1) when the FDC detects a CRC error in a sector data field. This flag is not set when a CRC error is detected in the ID field.
4	WC	Cylinder Address Error. Set when the cylinder address from the disk sector ID field is different from the current cylinder address maintained within the FDC.
3	SH	Scan Hit. Set during the execution of the Scan command if the scan condition is satisfied.
2	SN	Scan Not Satisfied. Set during execution of the Scan command if the FDC cannot locate a sector on the specified cylinder that satisfies the scan condition.

1	BC	Bad Track Error. Set when the cylinder address from the disk sector ID field is FF hexadecimal and this cylinder address is different from the current cylinder address maintained within the FDC. This all "ones" cylinder number indicates a bad track (one containing hard errors) according to the IBM soft-sectored format specifications.
0	MD	Missing Data Address Mark Error. Set if the FDC cannot detect a data address mark or deleted data address mark on the specified track.

Status Register 3

BIT NUMBER	SYMBOL	DESCRIPTION
7	FT	Fault. This flag indicates the status of the fault signal from the selected disk drive.
6	WP	Write Protected. This flag indicates the status of the write protect signal from the selected disk drive.
5	RDY	Ready. This flag indicates the status of the ready signal from the selected disk drive.
4	T0	Track 0. This flag indicates the status of the track 0 signal from the selected disk drive.
3	TS	Two-Sided. This flag indicates the status of the two-sided signal from the selected disk drive.
2	H	Head Address. This flag indicates the status of the side select signal for the currently selected disk drive.
1,0	DS1,DS0	Drive Select. Indicates the currently selected disk drive number.

Execution Phase

All data transfers to (or from) the floppy drive occur during the execution phase. The 8272 has two primary modes of operation for data transfers (selected by the specify command):

1) DMA mode
2) non-DMA mode

In the DMA mode, execution phase data transfers are handled by the DMA controller hardware (invisible to the driver software). The driver software, however, must set all appropriate DMA controller registers prior to the beginning of the disk operation. An interrupt is generated by the 8272 after the last data transfer, indicating the completion of the execution phase, and the beginning of the result phase.

In the non-DMA mode, transfer requests are indicated by generation of an interrupt and by activation of the RQM flag (bit 7 in the Main Status Register). The interrupt signal can be used for interrupt-driven systems and RQM can be used for polled systems. The driver software must respond to the transfer request by reading data from, or writing data to, the FDC. After completing the last transfer, the 8272 generates an interrupt to indicate the beginning of the result phase. In the non-DMA mode, the processor must activate the "terminal count" (TC) signal to the FDC (normally by means of an I/O port) after the transfer request for the last data byte has been received (by the driver) and before the appropriate data byte has been read from (or written to) the FDC.

In either mode of operation (DMA or non-DMA), the execution phase ends when a "terminal count" signal is sensed by the FDC, when the last sector on a track (the EOT parameter - Table 4) has been read or written, or when an error occurs.

Multi-sector and Multi-track Transfers

During disk read/write transfers (Read Data, Write Data, Read Deleted Data, and Write Deleted Data), the FDC will continue to transfer data from sequential sectors until the TC input is sensed. In the DMA mode, the TC input is normally set by the DMA controller. In the non-DMA mode, the processor directly controls the FDC TC input as previously described. Once the TC input is received, the FDC stops requesting data transfers (from the system software or DMA controller). The FDC, however, continues to read data from, or write data to, the floppy disk until the end of the current disk sector. During a disk read operation, the data read from the disk (after reception of the TC input) is discarded, but the data CRC is checked for errors; during a disk write operation, the remainder of the sector is filled with all-zero bytes.

If the TC signal is not received before the last byte of the current sector has been transferred to/from the system, the FDC increments the sector number by one and initiates a read or write command for this new disk sector.

The FDC is also designed to operate in a multi-track mode for dual-sided disks. In the multi-track mode (specified by means of the MT flag in the command byte - Table 4) the FDC will automatically increment the head address (from 0 to 1) when the last sector (on the track under head 0) has been read or written. Reading or writing is then continued on the first sector (sector 1) of head 1.

Drive Status Polling

After the power-on reset, the 8272 automatically enters a drive status polling mode. If a change in drive status is detected (all drives are assumed to be "not ready" at power-on), an interrupt is generated. The 8272 continues this status polling between command executions (and between step pulses in the Seek command). In this manner, the 8272 automatically notifies the system software whenever a floppy disk is inserted, removed, or changed by the operator.

Command Details

During the command phase, the Main Status Register must be polled by the driver software before each byte is written into the Data Register. The DIO (bit 6) and RQM (bit 7) flags in the Main Status Register must be low and high, respectively, before each byte of the command may be written into the 8272. The beginning of the execution phase for any of these commands will cause DIO to be set high and RQM to be set low.

Operation of the FDC commands is described in detail in Application Note AP-116, "An Intelligent Data Base System Using the 8272."

Invalid Commands

If an invalid (undefined) command is sent to the FDC, the FDC will terminate the command. No interrupt is generated by the 8272 during this condition. Bit 6 and bit 7 (DIO and RQM) in the Main Status Register are both set indicating to the processor that the 8272 is in the result phase and the contents of Status Register 0 must be read. When the processor reads Status Register 0 it will find an 80H code indicating that an invalid command was received. The driver software in Appendix B checks each requested command and will not issue an invalid command to the 8272.

A Sense Interrupt Status command must be sent after a Seek or Recalibrate interrupt; otherwise the FDC will consider the next command to be an invalid command. Also, when the last "hidden" interrupt has been serviced, further Sense Interrupt Status commands will result in invalid command codes.

4. 8272 Physical Interface Software

PL/M software driver listings for the 8272 FDC are contained in Appendix A. These drivers have been designed to operate in a DMA environment (as described in Application Note AP-116, "An Intelligent Data Base System Using the 8272"). In the following paragraphs, each driver procedure is described. (A description of the driver data base variables is given in Table 7.) In addition, the modifications necessary to reconfigure the drivers for operation in a polled environment are discussed.

INITIALIZE$DRIVERS

This initialization procedure must be called before any FDC operations are attempted. This module initializes the DRIVE$READY, DRIVE$STATUS$CHANGE, OPERATION$IN$PROGRESS, and OPERATION$COMPLETE arrays as well as the GLOBAL$DRIVE$NO variable.

EXECUTE$DOCB

This procedure contains the main 8272 driver control software and handles the execution of a complete FDC command. EXECUTE$DOCB is called with two parameters: a) a pointer to a disk operation control block and b) a pointer to a result status byte. The format of the disk operation control block is illustrated in Figure 2 and the result status codes are described in Table 8.

Before starting the command phase for the specified disk operation, the command is checked for validity and to determine whether the FDC is busy. (For an overlapped operation, if the FDC BUSY flag is set — in the Main Status Register — the command cannot be started; non-overlapped operations cannot be started if the FDC BUSY flag is set, if any drive is in the process of seeking/recalibrating, or if an operation is currently in progress on the specified drive.)

After these checks are made, interrupts are disabled in order to set the OPERATIONINPROGRESS flag, reset the OPERATION$COMPLETE flag, load a pointer to the current operation control block into the OPERATION$DOCB$PTR array and set GLOBAL$DRIVE$NO (if a non-overlapped operation is to be started).

At this point, parameters from the operation control block are output to the DMA controller and the FDC command phase is initiated. After completion of the command phase, a test is made to determine the type of result phase required for the current operation. If no result phase is needed, control is immediately returned to the calling program. If an immediate result phase is required, the result bytes are input from the FDC. Otherwise, the CPU waits until the OPERATION$COMPLETE flag is set (by the interrupt service procedure).

Finally, if an error is detected in the result status code (from the FDC), an FDC operation error is reported to the calling program.

Table 7: Driver Data Base

NAME	DESCRIPTION
DRIVE$READY	A public array containing the current "ready" status of each drive.
DRIVE$STATUS$CHANGE	A public array containing a flag for each drive. The appropriate flag is set whenever the ready status of a drive changes.
OPERATION$DOCB$PTR	An internal array of pointers to the operation control block currently in progress for each drive.
OPERATIONINPROGRESS	An internal array used by the driver procedures to determine if a disk operation is in progress on a given drive.
OPERATION$COMPLETE	An internal array used by the driver procedures to determine when the execution phase of a disk operation is complete.
GLOBAL$DRIVE$NO	A data byte that records the current drive number for non-overlapped disk operations.
VALID$COMMAND	A constant flag array that indicates whether a specified FDC command code is valid.
COMMAND$LENGTH	A constant byte array specifying the number of command/parameter bytes to be transferred to the FDC during the command phase.
DRIVENOPRESENT	A constant flag array that indicates whether a drive number is encoded into an FDC command.
OVERLAP$OPERATION	A constant flag array that indicates whether an FDC command can be overlapped with other commands.
NO$RESULT	A constant flag array that is used to determine when an FDC operation does not have a result phase.
IMMED$RESULT	A constant flag array that indicates that an FDC operation has a result phase beginning immediately after the command phase is complete.
POSSIBLE$ERROR	A constant flag array that indicates if an FDC operation should be checked for an error status indication during the result phase.

Address Offset	Disk Operation Control Block (DOCB)
0	DMA$OP
1	DMA$ADDR
3	DMA$ADDR$EXT
4	DMA$COUNT
6	DISK$COMMAND(0)
7	DISK$COMMAND(1)
8	DISK$COMMAND(2)
9	DISK$COMMAND(3)
10	DISK$COMMAND(4)
11	DISK$COMMAND(5)
12	DISK$COMMAND(6)
13	DISK$COMMAND(7)
14	DISK$COMMAND(8)
15	DISK$RESULT(0)
16	DISK$RESULT(1)
17	DISK$RESULT(2)
18	DISK$RESULT(3)
19	DISK$RESULT(4)
20	DISK$RESULT(5)
21	DISK$RESULT(6)
22	MISC

AFN-01949A

Figure 2. Disk Operation Control Block (DOCB) Format

Table 8: EXECUTE$DOCB Return Status Codes

Code	Description
0	No errors. The specified operation was completed without error.
1	FDC busy. The requested operation cannot be started. This error occurs if an attempt is made to start an operation before the previous operation is completed.
2	FDC error. An error was detected by the FDC during the execution phase of a disk operation. Additional error information is contained in the result data portion of the disk operation control block (DOCB.DISK$RESULT) as described in the 8272 data sheet. This error occurs whenever the 8272 reports an execution phase error (e.g., missing address mark).
3	8272 command interface error. An 8272 interfacing error was detected during the command phase. This error occurs when the command phase of a disk operation cannot be successfully completed (e.g., incorrect setting of the DIO flag in the Main Status Register).
4	8272 result interface error. An 8272 interfacing error was detected during the result phase. This error occurs when the result phase of a disk operation cannot be successfully completed (e.g., incorrect setting of the DIO flag in the Main Status Register).
5	Invalid FDC Command.

FDCINT

This procedure performs all interrupt processing for the 8272 interface drivers. Basically, two types of interrupts are generated by the 8272: (a) an interrupt that signals the end of a command execution phase and the beginning of the result phase and (b) an interrupt that signals the completion of an overlapped operation or the occurrence of an unexpected event (e.g., change in the drive "ready" status).

An interrupt of type (a) is indicated when the FDC BUSY flag is set (in the Main Status Register). When a type (a) interrupt is sensed, the result bytes are read from the 8272 and placed in the result portion of the disk operation control block, the appropriate OPERATION$COMPLETE flag is set, and the OPERATIONINPROGRESS flag is reset.

When an interrupt of type (b) is indicated (FDC not busy), a sense interrupt status command is issued (to the FDC). The upper two bits of the result status register (Status Register Zero - ST0) are used to determine the cause of the interrupt. The following four cases are possible:

1) Operation Complete. An overlapped operation is complete. The drive number is found in the lower two bits of ST0. The ST0 data is transferred to the active operation control block, the OPERATION$COMPLETE flag is set, and the OPERATIONINPROGRESS flag is reset.

2) Abnormal Termination. A disk operation has abnormally terminated. The drive number is found in the lower two bits of ST0. The ST0 data is transferred to the active control block, the OPERATION$COMPLETE flag is set, and the OPERATIONINPROGRESS flag is reset.

3) Invalid Command. The execution of an invalid command (i.e., a sense interrupt command with no interrupt pending) has been attempted. This interrupt signals the successful completion of all interrupt processing.

4) Drive Status Change. A change has occurred in the "ready" status of a disk drive. The drive number is found in the lower two bits of ST0. The DRIVE$READY flag for this disk drive is set to the new drive "ready" status and the DRIVE$STATUS$CHANGE flag for the drive is also set. In addition, if a command is currently in progress, the ST0 data is transferred to the active control block, the OPERATION$COMPLETE flag is set, and the OPERATIONINPROGRESS flag is reset.

After processing a type (b) interrupt, additional sense interrupt status commands must be issued and processed until an "invalid command" result is returned from the FDC. This action guarantees that all "hidden" interrupts are serviced.

In addition to the major driver procedures described above, a number of support procedures are required. These support routines are briefly described in the following paragraphs.

OUTPUT$CONTROLS$TO$DMA

This procedure outputs the DMA mode, the DMA address, and the DMA word count to the 8237 DMA controller. In addition, the upper four bits of the 20-bit DMA address are output to the address extension latch. Finally, the disk DMA channel is started.

OUTPUT$COMMAND$TO$FDC

This software module outputs a complete disk command to the 8272 FDC. The number of required command/parameter bytes is found in the COMMAND$LENGTH table. The appropriate bytes are output one at a time (by calls to OUTPUT$BYTE$TO$FDC) from the command portion of the disk operation control block.

INPUT$RESULT$FROM$FDC

This procedure is used to read result phase status information from the disk controller. At most, seven bytes are read. In order to read each byte, a call is made to INPUT$BYTE$FROM$FDC. When the last byte has been read, a check is made to insure that the FDC is no longer busy.

OUTPUT$BYTE$TO$FDC

This software is used to output a single command/parameter byte to the FDC. This procedure waits until the FDC is ready for a command byte and then outputs the byte to the FDC data port.

INPUT$BYTE$FROM$FDC

This procedure inputs a single result byte from the FDC. The software waits until the FDC is ready to transfer a result byte and then reads the byte from the FDC data port.

FDC$READY$FOR$COMMAND

This procedure assures that the FDC is ready to accept a command/parameter byte by performing the following three steps. First, a small time interval (more than 20 microseconds) is inserted to assure that the RQM flag has time to become valid (after the last byte transfer). Second, the master request flag (RQM) is polled until it is activated by the FDC. Finally, the DIO flag is checked to ensure that it is properly set for FDC input (from the processor).

FDC$READY$FOR$RESULT

The operation of this procedure is similar to the FDC$READY$FOR$COMMAND with the following exception. If the FDC BUSY flag (in the Main Status Register) is not set, the result phase is complete and no more data is available from the FDC. Otherwise, the procedure waits for the RQM flag and checks the DIO flag for FDC output (to the processor).

OPERATION$CLEAN$UP

This procedure is called after the execution of a disk operation that has no result phase. OPERATION$CLEAN$UP resets the OPERATIONINPROGRESS flag and the GLOBAL$DRIVE$NO variable if appropriate. This procedure is also called to clean up after some disk operation errors.

Modifications for Polling Operation

To operate in the polling mode, the following modifications should be made to the previous routines:

1. The OUTPUT$CONTROLS$TO$DMA routine should be deleted.

2. In EXECUTE$DOCB, immediately prior to WAITFOROP$COMPLETE, a polling loop should be inserted into the code. The loop should test the RQM flag (in the Main Status Register). When RQM is set, a data byte should be written to, or read from, the 8272. The buffer address may be computed from the base address contained in DOCB.DMA$ADDR and DOCB.DMA$ADDR$EXT. After the correct number of bytes have been transferred, an operation complete interrupt will be issued by the FDC. During data transfer in the non-DMA mode, the NON-DMA MODE flag (bit 5 of the Main Status Register) will be set. This flag will remain set for the complete execution phase. When the transfer is finished, the NON-DMA MODE flag is reset and the result phase interrupt is issued by the FDC.

5. 8272 Logical Interface Software

Appendix B of this Application Note contains a PL/M listing of an exerciser program for the 8272 drivers. This program illustrates the design of logical interface level procedures to specify disk parameters, recalibrate a drive, seek to a cylinder, format a disk, read data, and write data.

The exerciser program is written to operate a standard single-sided 8" floppy disk drive in either the single- or double-density recording mode. Only the eight parameters listed in Table 9 must be specified. All other parameters are derived from these 8 basic variables.

Each of these logical interface procedures is described in the following paragraphs (refer to the listing in Appendix B).

SPECIFY

This procedure sets the FDC signal timing so that the FDC will interface correctly to the attached disk drive. The SPECIFY procedure requires four parameters, the step rate (SRT), head load time (HLT), head unload time (HUT), and the non-DMA mode flag (ND). This procedure builds a disk operation control block (SPECIFY$DOCB) and passes the control block to the FDC driver module (EXECUTE$DOCB) for execution. (Note carefully the computation required to transform the step rate (SRT) into the correct 8272 parameter byte.)

RECALIBRATE

This procedure causes the floppy disk read/write head to retract to track 0. The RECALIBRATE procedure requires only one parameter — the drive number on which the recalibrate operation is to be performed. This procedure builds a disk operation control block (RECALIBRATE$DOCB) and passes the control block to the FDC driver for execution.

SEEK

This procedure causes the disk read/write head (on the selected drive) to move to the desired cylinder position. The SEEK procedure is called with three parameters: drive number (DRV), head/side number (HD), and cylinder number (CYL). This software module builds a disk operation control block (SEEK$DOCB) that is executed by the FDC driver.

FORMAT

The FORMAT procedure is designed to initialize a complete floppy disk so that sectors can subsequently be read and written by system and application programs. Three parameters must be supplied to this procedure: the drive number (DRV), the recording density (DENS), and the interleave factor (INTLVE). The FORMAT procedure generates a data block (FMTBLK) and a disk operation control block (FORMAT$DOCB) for each track on the floppy disk (normally 77).

Table 9: Basic Disk Parameters

Name	Description
DENSITY	The recording mode (FM or MFM).
FILLER$BYTE	The data byte to be written in all sectors during formatting.
TRACKSPERDISK	The number of cylinders on the floppy disk.
BYTESPERSECTOR	The number of bytes in each disk sector. The exerciser accepts 128, 256, and 512 in FM mode, and 256, 512, and 1024 in MFM mode.
INTERLEAVE	The sector interleave factor for each disk track.
STEP$RATE	The disk drive step rate (1-16 milliseconds).
HEAD$LOAD$TIME	The disk drive head load time (2-254 milliseconds).
HEAD$UNLOAD$TIME	The head unload time (16-240 milliseconds).

The format data block specifies the four sector ID field parameters (cylinder, head, sector, and bytes per sector) for each sector on the track. The sector numbers need not be sequential; the interleave factor (INTLVE parameter) is used to compute the logical to physical sector mapping.

After both the format data block and the operation control block are generated for a given cylinder, control is passed to the 8272 drivers for execution. After the format operation is complete, a SEEK to the next cylinder is performed, a new format table is generated, and another track formatting operation is executed by the drivers. This track formatting continues until all tracks on the diskette are formatted.

In some systems, bad tracks must also be specified when a disk is formatted. For these systems, the existing FORMAT procedure should be modified to format bad tracks with a cylinder number of OFFH.

WRITE

The WRITE procedure transfers a complete sector of data to the disk drive. Five parameters must be supplied to this software module: the drive number (DRV), the cylinder number (CYL), the head/side number (HD), the sector number (SEC) and the recording density (DENS). This procedure generates a disk operation control block (WRITE$DOCB) from these parameters and passes the control block to the 8272 driver for execution. When control returns to the calling program, the data has been transferred to disk.

READ

This procedure is identical to the WRITE procedure except the direction of data transfer is reversed. The READ procedure transfers a sector of data from the floppy disk to system memory.

Coping With Errors

In actual practice all logical disk interface routines would contain error processing mechanisms. (Errors have been ignored for the sake of simplicity in the exerciser programs listed in Appendix B.) A typical error recovery technique consists of a two-stage procedure. First, when an error is detected, a recalibrate operation is performed followed by a retry of the failed operation. This procedure forces the drive to seek directly to the requested cylinder (lowering the probability of a seek error) and attempts to perform the requested operation an additional time. Soft (temporary) errors caused by mechanical or electrical interference do not normally recur during the retry operation; hard errors (caused by media or drive failures), on the other hand, will continue to occur during retry operations. If, after a number of retries (approximately 10), the operation continues to fail, an error message is displayed to the system operator. This error message lists the drive number, type of operation, and failure status (from the FDC). It is the operator's responsibility to take additional action as required.

6. File Systems

The file system provides the disk I/O interface level most familiar to users of interactive microcomputer and minicomputer systems. In a file system, all data is stored in named disk areas called files. The user and applications programs need not be concerned with the exact location of a file on the disk — the disk file system automatically determines the file location from the file name. Files may be created, read, written, modified, and finally deleted (destroyed) when they are no longer needed. Each floppy disk typically contains a directory that lists all the files existing on the disk. A directory entry for a file contains information such as file name, file size, and the disk address (track and sector) of the beginning of the file.

File Allocation

File storage is actually allocated on the disk (by the file system) in fixed size areas called blocks. Normally a block is the same size as a disk sector. Files are created by finding and reserving enough unused blocks to contain the data in the file. Two file allocation methods are currently in widespread use. The first method allocates blocks (for a file) from a sequential pool of unused blocks. Thus, a file is always contained in a set of sequential blocks on the disk. Unfortunately, as files are created, updated, and deleted, these free-block pools become fragmented (separated from one another). When this fragmentation occurs, it often becomes impossible for the file system to create a file even though there is a sufficient number of free blocks on the disk. At this point, special programs must be run to "squeeze" or compact the disk, in order to re-create a single contiguous free-block pool.

The second file allocation method uses a more flexible technique in which individual data blocks may be located anywhere on the disk (with no restrictions). With this technique, a file directory entry contains the disk address of a file pointer block rather than the disk address of the first data block of the file. This file pointer block contains pointers (disk addresses) for each data block in the file. For example, the first pointer in the file pointer block contains the track and sector address of the first data block in the file, the second pointer contains the disk address of the second data block, etc.

In practice, pointer blocks are usually the same size as data blocks. Therefore, some files will require multiple pointer blocks. To accommodate this requirement without loss of flexibility, pointer blocks are linked together, that is, each pointer block contains the disk address of the following pointer block. The last pointer block of the file is signalled by an illegal disk address (e.g., track 0, sector 0 or track OFFH, sector OFFH).

The Intel File System

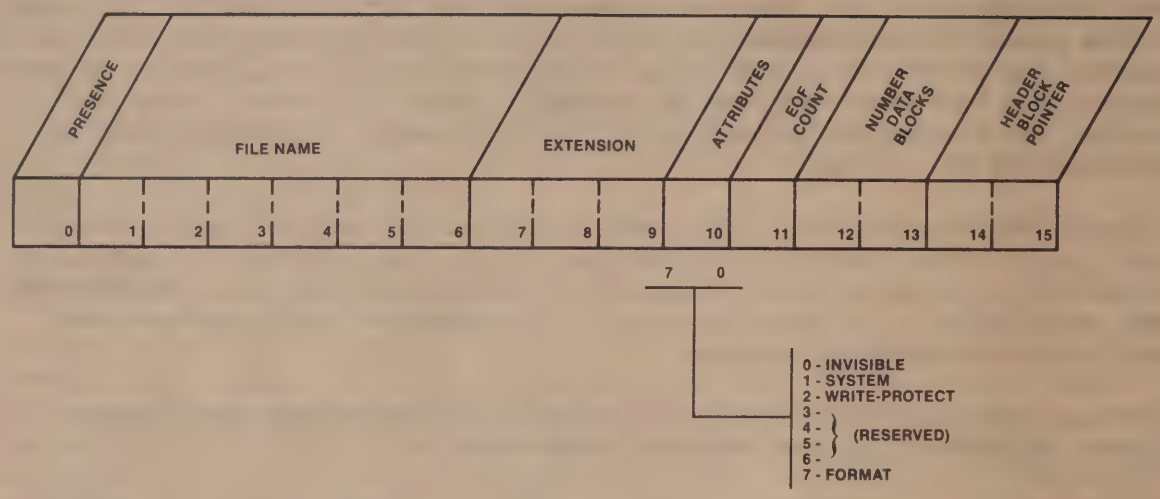
The Intel file system (described in detail in the RMX-80 Users Guide) uses the second disk file allocation method (previously discussed). In order to lower the system overhead involved in finding free data blocks, the Intel file system incorporates a free space management data structure known as a bit map. Each disk sector is represented by a single bit in the bit map. If a bit in the bit map is set to 1, the corresponding disk sector has been allocated. A zero in the bit map indicates that the corresponding sector is free. With this technique, the process of allocating or freeing a sector is accomplished by simply altering the bit map.

File names consist of a basic file name (up to six characters) and a file extension (up to three characters). The basic file name and the file extension are separated by a period (.). Examples of valid file names are: DRIV72.OBJ, XX.TMP, and FILE.CS. In addition, four file attributes are supported (see Figure 3 for attribute definitions).

The bit map and the file directory are placed on prespecified disk tracks (reserved for system use) beginning at track zero.

Disk File System Functions

Table 2 illustrates the typical functions implemented by a disk file system. As an example, the disk directory function (DIR) lists disk file information on the console display terminal. Figure 3 details the contents of a display entry in the Intel file system. The PL/M procedure outlined in Figure 4 illustrates a disk directory algorithm that displays the file name, the file attributes, and the file size (in blocks) for each file in the directory.



Directory Entry

Presence is a flag that can contain one of three values:

000H - The file associated with this entry is present on the disk.

07FH - No file is associated with this entry; the content of the rest of the entry is undefined. The first entry with its flag set to 07FH marks the current logical end of the directory and directory searches stop at this entry.

0FFH - The file named in this entry once existed on the disk but is currently deleted. The next file added to the directory will be placed in the first entry marked 0FFH. This flag cannot, therefore, be used to (reliably) find a file that has been deleted. A value of 0FFH should be thought of as simply marking an open directory entry.

File Name is a string of up to 6 non-blank ASCII characters specifying the name of the file associated with the directory entry. If the file name is shorter than six characters, the remaining bytes contain binary zeros. For example, the name ALPHA would be stored as: 414C50484100H.

Extension is a string of up to 3 non-blank ASCII characters that specifies an extension to the file name. Extensions often identify the type of data in the file such as OBJ (object module), or PLM (PL/M source module). As with the file name, unused positions in the extension field are filled with binary zeros.

Figure 3. Intel Directory Entry Format

Attributes are bits that identify certain characteristics of the file. A 1 bit indicates that the file has the attribute, while a 0 bit means that the file does not have the attribute. The bit positions and their corresponding attributes are listed below (bit 0 is the low-order or rightmost bit, bit 7 is the leftmost bit):

0: Invisible. Files with this attribute are not listed by the ISIS-II DIR command unless the I switch is used. All system files are invisible.

1: System. Files with this attribute are copied to the disk in drive 1 when the S switch is specified with the ISIS-II FORMAT command.

2: Write-Protect. Files with this attribute cannot be opened for output or update, nor can they be deleted or renamed.

3-6: These positions are reserved for future use.

7: Format. Files with this attribute are treated as though they are write-protected. In addition, these files are created on a new diskette when the ISIS-II FORMAT command is issued. The system files all have the FORMAT attribute and it should not be given to any other files.

EOF Count contains the number of the last byte in the last data block of the file. If the value of this field is 080H, for example, the last byte in the file is byte number 128 in the last data block (the last block is full).

Number of Data Blocks is an address variable that indicates the number of data blocks currently used by the file. ISIS-II and the RMX/80 Disk File system both maintain a counter called LENGTH that is the current number of bytes in the file. This is calculated as:

((NUMBER OF DATA BLOCKS - 1) x 128 + EOF COUNT.

Header Block Pointer is the address of the file's header block. The high byte of the field is the sector number and the low byte is the track number. The system "finds" a disk file by searching the directory for the name and then using the header block pointer to seek to the beginning of the file.

Figure 3. Intel Directory Entry Format (Continued)

```
dir: procedure(drv,dens)  public;
  declare  drv            byte,
           dens           byte,
           sector         byte,
           i              byte,
           dir$ptr        byte,
           dir$entry      based rdbptr structure (presence byte,
                          file$name(6) byte,extension(3) byte,
                          attribute byte,eof$count byte,
                          data$blocks address,header$ptr address),
           size (5)       byte,

           invisible$flag literally '1',
           system$flag    literally '2',
           protected$flag literally '4',
           format$flag    literally '80H';


  /* The disk directory starts at cylinder 1, sector 2 */
  call seek(drv,1,0);
  do sector=2 to 26;
    call read(drv,1,0,sector,dens);
    do dir$ptr=0 to 112 by 4;
      if dir$entry.presence=7FH then return;
      if dir$entry.presence=0
        then do;
          do i=0 to 5; call co(dir$entry.file$name(i)); end;
          call co(period);
          do i=0 to 2; call co(dir$entry.extension(i)); end;
          do i=0 to 4; call co(space); end;
          call convert$to$decimal(@size,dir$entry.data$blocks);
          do i=0 to 4; call co(size(i)); end;
          If (dir$entry.attribute and invisible$flag) <> 0 then call co('I');
          If (dir$entry.attribute and system$flag) <> 0 then call co('S');
          If (dir$entry.attribute and protected$flag) <> 0 then call co('W');
          If (dir$entry.attribure and format$flag) <> 0 then call co('F');
        end;
    end;
  end;

end dir;
```

AFN-01949A

Figure 4. Sample PL/M Directory Procedure

7. Key 8272 Software Interfacing Considerations

This section contains a quick review of Key 8272 Software design features and issues. (Most items have been mentioned in other sections of this application note.) Before designing 8272 software drivers, it is advisable that the information in this section be thoroughly understood.

1. Non-DMA Data Transfers

In systems that operate without a DMA controller (in the polled or interrupt driven mode), the system software is responsible for counting data transfers to/from the 8272 and generating a TC signal to the FDC when the transfer is complete.

2. Processor Command/Result Phase Interface

In the command phase, the driver software must write the exact number of parameters in the exact order shown in Table 5. During the result phase, the driver must read the complete result status. For example, the Format Track command requires six command bytes and presents seven result bytes. The 8272 will not accept a new command until all result bytes are read. Note that the number of command and result bytes varies from command-to-command. **Command and result phases cannot be shortened.**

During both the command and result phases, the Main Status Register must be read by the driver before each byte of information is read from, or written to, the FDC Data Register. Before each command byte is written, DIO (bit 6) must be low (indicating a data transfer from the processor) and RQM (bit 7) must be high (indicating that the FDC is ready for data). During the result phase, DIO must be high (indicating a data transfer to the processor) and RQM must also be high (indicating that data is ready for the processor).

Note: After the 8272 receives a command byte, the RQM flag may remain set for approximately 16 microseconds (with an 8 MHz clock). The driver should not attempt to read the Main Status Register before this time interval has elapsed; otherwise, the driver may erroneously assume that the FDC is ready to accept the next byte.

3. Sector Sizes

The 8272 does not support 128 byte sectors in the MFM (double-density) mode.

4. Drive Status Changes

The 8272 constantly polls all drives for changes in the drive ready status. This polling begins immediately following RESET. An interrupt is generated every time the FDC senses a change in the drive ready status. After reset, the FDC assumes that all drives are "not ready". If a drive is ready immediately after reset, the 8272 generates a drive status change interrupt.

5. Seek Commands

The 8272 FDC does not perform implied seeks. Before issuing a data read or write command, the read/write head must be positioned over the correct cylinder by means of an explicit seek command. If the head is not positioned correctly, a cylinder address error is generated.

6. Interrupt Processing

When the processor receives an interrupt from the FDC, the FDC may be reporting one of two distinct events:

a) The beginning of the result phase of a previously requested read, write, or scan command.

b) An asynchronous event such as a seek/recalibrate completion, an attention, an abnormal command termination, or an invalid command.

These two cases are distinguished by the FDC BUSY flag (bit 4) in the Main Status Register. If the FDC BUSY flag is high, the interrupt is of type (a). If the FDC BUSY flag is low, the interrupt was caused by an asynchronous event (b).

A single interrupt from the FDC may signal more than one of the above events. After receiving an interrupt, the processor must continue to issue Sense Interrupt Status commands (and service the resulting conditions) until an invalid command code is received. In this manner, all "hidden" interrupts are ferreted out and serviced.

7. Skip Flag (SK)

The skip flag is used during the execution of Read Data, Read Deleted Data, Read Track, and various Scan commands. This flag permits the FDC to skip unwanted sectors on a disk track.

When performing a Read Data, Read Track, or Scan command, a high SK flag indicates that the FDC is to skip over (not transfer) any sector containing a deleted data address mark. A low SK flag indicates that the FDC is to terminate the command (after reading all the data in the sector) when a deleted data address mark is encountered.

When performing a Read Deleted Data command, a high SK flag indicates that sectors containing normal data address marks are to be skipped. Note that this is just the opposite situation from that described in the last paragraph. When a data address mark is encountered during a Read Deleted Data command (and the SK flag is low), the FDC terminates the command after reading all the data in the sector.

8. Bad Track Maintenance

The 8272 does not internally maintain bad track information. The maintenance of this information must be performed by system software. As an example of typical bad track operation, assume that a media test determines that track 31 and track 66 of a given floppy disk are bad. When the disk is formatted for use, the system software formats physical track 0 as logical cylinder 0 (C=0 in the command phase parameters), physical track 1 as logical track 1 (C=1), and so on, until physical track 30 is formatted as logical cylinder 30 (C=30). Physcial track 31 is bad and should be formatted as logical cylinder FF (indicating a bad track). Next, physical track 32 is formatted as logical cylinder 31, and so on, until physical track 65 is formatted as logical cylinder 64. Next, bad physical track 66 is formatted as logical cylinder FF (another bad track marker), and physical track 67 is formatted as logical cylinder 65. This formatting continues until the last physical track (77) is formatted as logical cylinder 75. Normally, after this formatting is complete, the bad track information is stored in a prespecified area on the floppy disk (typically in a sector on track 0) so that the system will be able to recreate the bad track information when the disk is removed from the drive and reinserted at some later time.

To illustrate how the system software performs a transfer operation on a disk with bad tracks, assume that the disk drive head is positioned at track 0 and the disk described above is loaded into the drive. If a command to read track 36 is issued by an application program, the system software translates this read command into a **seek** to physical track 37 (since there is one bad track between 0 and 36, namely 31) followed by a **read** of logical cylinder 36. Thus, the cylinder parameter C is set to 37 for the Seek command and 36 for the Read Sector command.

REFERENCES

1. Intel, "8272 Single/Double Density Floppy Disk Controller Data Sheet," Intel Corporation, 1980.

2. Intel, "An Intelligent Data Base System Using the 8272," Intel Application Note, AP-116, 1981.

3. Intel, iSBC 208 Hardware Reference Manual, Manual Order No. 143078, Intel Corporation, 1980.

4. Intel, RMX/80 User's Guide, Manual Order No. 9800522, Intel Corporation, 1978

5. Brinch Hansen, P., Operating System Principles, Prentice-Hall, Inc., New Jersey, 1973.

6. Flores, I., Computer Software: Programming Systems for Digital Computers, Prentice-Hall, Inc., New Jersey, 1965.

7. Knuth, D. E., Fundamental Algorithms, Addison-Wesley Publishing Company, Massachusetts, 1975.

8. Shaw, A. C., The Logical Design of Operating Systems, Prentice-Hall, Inc., New Jersey, 1974.

9. Watson, R. W., Time Sharing System Design Concepts, McGraw-Hill, Inc., New York, 1970.

10. Zarrella, J., Operating Systems: Concepts and Principles, Microcomputer Applications, California, 1979.

APPENDIX A
8272 FDC DEVICE DRIVER SOFTWARE

```
PL/M-86 COMPILER    8272 FLOPPY DISK CONTROLLER DEVICE DRIVERS

ISIS-II PL/M-86 V1.2 COMPILATION OF MODULE DRIVERS
OBJECT MODULE PLACED IN :F1:driv72.OBJ
COMPILER INVOKED BY:  plm86 :F1:driv72.p86 DEBUG

             $title('8272 floppy disk controller device drivers')
             $nointvector
             $optimize(2)
             $large

  1          drivers: do;

  2   1      declare
               /* floppy disk port definitions */
               fdc$status$port        literally '30H',        /* 8272 status port */
               fdc$data$port          literally '31H';        /* 8272 data port */

  3   1      declare
              /* floppy disk commands */
               sense$int$status       literally '08H';

  4   1      declare
               /* interrupt definitions */
               fdc$int$level          literally '33';     /* fdc interrupt level */

  5   1      declare
               /* return status and error codes */
               error                  literally '0',
               ok                     literally '1',
               complete               literally '3',
               false                  literally '0',
               true                   literally '1',
               error$in               literally 'not',
               propagate$error        literally 'return error',

               stat$ok                literally '0',          /* fdc operation completed without errors */
               stat$busy              literally '1',          /* fdc is busy, operation cannot be started */
               stat$error             literally '2',          /* fdc operation error */
               stat$command$error     literally '3',          /* fdc not ready for command phase */
               stat$result$error      literally '4',          /* fdc not ready for result phase */
               stat$invalid           literally '5';          /* invalid fdc command */

  6   1      declare
               /* masks */
               busy$mask              literally '10H',
               DIO$mask               literally '40H',
               RQM$mask               literally '80H',
               seek$mask              literally '0FH',
               result$error$mask      literally '0C0H',
               result$drive$mask      literally '03H',
               result$ready$mask      literally '08H';

  7   1      declare
               /* drive numbers */
               max$no$drives          literally '3',
               fdc$general            literally '4';

  8   1      declare
               /* miscellaneous control */
               any$drive$seeking      literally '((input(fdc$status$port) and seek$mask) <> 0)',
               command$code           literally '(docb.disk$command(0) and 1FH)',
               DIO$set$for$input      literally '((input(fdc$status$port) and DIO$mask)=0)',
               DIO$set$for$output     literally '((input(fdc$status$port) and DIO$mask)<>0)',
               extract$drive$no       literally '(docb.disk$command(1) and 03H)',
               fdc$busy               literally '((input(fdc$status$port) and busy$mask) <> 0)',
               no$fdc$error           literally 'possible$error(command$code) and ((docb.disk$result(0)
                                        and result$error$mask) = 0)',
               wait$for$op$complete   literally 'do while not operation$complete(drive$no); end',
               wait$for$RQM           literally 'do while (input(fdc$status$port) and RQM$mask) = 0; end;';

  9   1      declare
               /* structures */
               docb$type              literally               /* disk operation control block */
                     '(dma$op byte,dma$addr word, dma$addr$ext byte,dma$count word,
                      disk$command(9) byte,disk$result(7) byte,misc byte)';

             $eject
 10   1      declare
               drive$status$change(4) byte public,            /* when set - indicates that drve status changed *
               drive$ready(4) byte public;                    /* current status of drives */
```

```
11  1   declare
          operation$in$progress(5) byte,                /* internal flags for operation with multiple drives */
          operation$complete(5) byte,                   /* fdc execution phase completed */
          operation$docb$ptr(5) pointer,                /* pointers for operations in progress */
          interrupt$docb structure docb$type,           /* temporary docb for interrupt processing */
          global$drive$no byte;                         /* drive number of non-overlapped operation
                                                           in progress - if any */

12  1   declare
          /* internal vectors that contain command operational information */
          no$result(32) byte                            /* no result phase to command */
            data(0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
          immed$result(32) byte                         /* immediate result phase for command */
            data(0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
          overlap$operation(32) byte                    /* command permits overlapped operation of drvies */
            data(0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
          drive$no$present(32) byte                     /* drive number present in command information */
            data(0,0,1,0,1,1,1,1,0,1,1,0,1,1,0,1,0,1,0,0,0,0,0,0,1,0,0,0,1,0,0),
          possible$error(32) byte                       /* determines if command can return with an error */
            data(0,0,1,0,0,1,1,1,1,1,1,0,1,1,0,1,0,1,0,0,0,0,0,0,0,1,0,0,0,1,0,0),
          command$length(32) byte                       /* contains number of command bytes for each command */
            data(0,0,9,3,2,9,9,2,1,9,2,0,9,6,0,3,0,9,0,0,0,0,0,0,0,9,0,0,0,9,0,0),
          valid$command(32) byte                        /* flags invalid command codes */
            data(0,0,1,1,1,1,1,1,1,1,1,0,1,1,0,1,0,1,0,0,0,0,0,0,0,1,0,0,0,1,0,0);

        $eject

        /**** initialization for the 8272 fdc driver software.  This procedure must
              be called prior to execution of any driver software.    ****/

13  1   initialize$drivers: procedure public;
        /* initialize 8272 drivers */
14  2     declare drv$no byte;

15  2     do drv$no=0 to max$no$drives;
16  3       drive$ready(drv$no)=false;
17  3       drive$status$change(drv$no)=false;
18  3       operation$in$progress(drv$no)=false;
19  3       operation$complete(drv$no)=false;
20  3     end;

21  2     operation$in$progress(fdc$general)=false;
22  2     operation$complete(fdc$general)=false;
23  2     global$drive$no=0;

24  2   end initialize$drivers;


        /**** wait until the 8272 fdc is ready to receive command/parameter bytes
              in the command phase.  The 8272 is ready to receive command bytes
              when the RQM flag is high and the DIO flag is low.    ****/

25  1   fdc$ready$for$command: procedure byte;

          /* wait for valid flag settings in status register */
26  2     call time(1);

          /* wait for "master request" flag */
27  2     wait$for$RQM;

          /* check data direction flag */
30  2     if DIO$set$for$input
            then return ok;
32  2       else return error;

33  2   end fdc$ready$for$command;


        /**** wait until the 8272 fdc is ready to return data bytes in the result
              phase.  The 8272 is ready to return a result byte when the RQM and DIO
              flags are both high.  The busy flag in the main status register will
              remain set until the last data byte of the result phase has been read
              by the processor.    ****/

34  1   fdc$ready$for$result: procedure byte;

          /* wait for valid settings in status register */
35  2     call time(1);

          /* result phase has ended when the 8272 busy flag is reset */
36  2     if not fdc$busy
            then return complete;
```

```
                /* wait for "master request" flag */
38    2         wait$for$RQM;

                /* check data direction flag */
41    2         if DIO$set$for$output
                  then return ok;
43    2           else return error;

44    2       end fdc$ready$for$result;


              /**** output a single command/parameter byte to the 8272 fdc.  The "data$byte"
                    parameter is the byte to be output to the fdc.    ****/

45    1       output$byte$to$fdc: procedure(data$byte) byte;
46    2         declare data$byte byte;

                /* check to see if fdc is ready for command */
47    2         if not fdc$ready$for$command
                  then propagate$error;

49    2         output(fdc$data$port)=data$byte;

50    2         return ok;
51    2       end output$byte$to$fdc;


              /**** input a single result byte from the 8272 fdc.  The "data$byte$ptr"
                    parameter is a pointer to the memory location that is to contain
                    the input byte.    ****/

52    1       input$byte$from$fdc: procedure(data$byte$ptr) byte;
53    2         declare data$byte$ptr pointer;
54    2         declare
                  data$byte based data$byte$ptr byte,
                  status byte;

                /* check to see if fdc is ready */
55    2         status=fdc$ready$for$result;
56    2         if error$in status
                  then propagate$error;

                /* check for result phase complete */
58    2         if status=complete
                  then return complete;

60    2         data$byte=input(fdc$data$port);
61    2         return ok;
62    2       end input$byte$from$fdc;

              $eject

              /**** output the dma mode, the dma address, and the dma word count to the
                    8237 dma controller.  Also output the high order four bits of the
                    address to the address extension latch.  Finally, start the disk
                    dma channel.  The "docb$ptr" parameter is a pointer to the appropriate
                    disk operation control block.    ****/

63    1       output$controls$to$dma: procedure(docb$ptr);
64    2         declare docb$ptr pointer;
65    2         declare docb based docb$ptr structure docbtype;

66    2         declare
                  /* dma port definitions */
                  dma$upper$addr$port   literally '10H',        /* upper 4 bits of current address */
                  dma$disk$addr$port    literally '00H',        /* current address port */
                  dma$disk$word$count   literally '01H',        /* word count port */
                  dma$command$port      literally '08H',        /* command port */
                  dma$mode$port         literally '0BH',        /* mode port */
                  dma$mask$sr$port      literally '0AH',        /* mask set/reset port */
                  dma$clear$ff$port     literally '0CH',        /* clear first/last flip-flop port */
                  dma$master$clear$port literally '0DH',        /* dma master clear port */
                  dma$mask$port         literally '0FH',        /* parallel mask set port*/

                  dma$disk$chan$start   literally '00H',        /* dma mask to start disk channel */
                  dma$extended$write    literally 'shl(1,5)',   /* extended write flag */
                  dma$single$transfer   literally 'shl(1,6)';   /* single transfer flag */

67    2         if docb.dma$op < 3
                  then do;
                    /* set dma mode and clear first/last flip-flop */
69    3           output(dma$mode$port)=shl(docb.dma$op,2) or 40H;
70    3           output(dma$clear$ff$port)=0;
```

```
                /* set dma address */
 71   3         output(dma$disk$addr$port)=low(docb.dma$addr);
 72   3         output(dma$disk$addr$port)=high(docb.dma$addr);
 73   3         output(dma$upper$addr$port)=docb.dma$addr$ext;

                /* output disk transfer word count to dma controller */
 74   3         output(dma$disk$word$count)=low(docb.dma$count);
 75   3         output(dma$disk$word$count)=high(docb.dma$count);

                /* start dma channel 0 for fdc */
 76   3         output(dma$mask$sr$port)=dma$disk$chan$start;
 77   3       end;

 78   2     end output$controls$to$dma;


            /**** output a high-level disk command to the 8272 fdc.  The number of bytes
                  required for each command is contained in the "command$length" table.
                  The "docb$ptr" parameter is a pointer to the appropriate disk operation
                  control block.    ****/

 79   1     output$command$to$fdc: procedure(docb$ptr) byte;
 80   2       declare docb$ptr pointer;

 81   2       declare
                docb based docb$ptr structure docb$type,
                cmd$byte$no byte;

 82   2       disable;

               /* output all command bytes to the fdc */
 83   2       do cmd$byte$no=0 to command$length(command$code)-1;
 84   3         if error$in output$byte$to$fdc(docb.disk$command(cmd$byte$no))
                  then do; enable; propagate$error; end;
 89   3       end;

 90   2       enable;
 91   2       return ok;
 92   2     end output$command$to$fdc;


            /**** input the result data from the 8272 fdc during the result phase (after
                  command execution).  The "docb$ptr" parameter is a pointer to the
                  appropriate disk operation control block.    ****/

 93   1     input$result$from$fdc: procedure(docb$ptr) byte;
 94   2       declare docb$ptr pointer;
 95   2       declare
                docb based docb$ptr structure docb$type,
                result$byte$no byte,
                temp byte,
                status byte;

 96   2       disable;

 97   2       do result$byte$no=0 to 7;
 98   3         status=input$byte$from$fdc(@temp);
 99   3         if error$in status
                  then do; enable; propagate$error; end;
104   3         if status=complete
                  then do; enable; return ok; end;
109   3         docb.disk$result(result$byte$no)=temp;
110   3       end;

111   2       enable;
112   2       if fdc$busy
                then return error;
114   2         else return ok;
115   2     end input$result$from$fdc;


            /**** cleans up after the execution of a disk operation that has no result
                  phase.  The procedure is also used after some disk operation errors.
                  "drv" is the drive number, and "cc" is the command code for the
                  disk operation.    ****/

116   1     operation$clean$up: procedure(drv,cc);
117   2       declare (drv,cc) byte;

118   2       disable;
119   2       operation$in$progress(drv)=false;
```

```
120   2        if not overlap$operation(cc)
                 then global$drive$no=0;
122   2        enable;

123   2      end operation$clean$up;

             $eject

             /**** execute the disk operation control block specified by the pointer
                   parameter "docb$ptr".  The "status$ptr" parameter is a pointer to
                   a byte variable that is to contain the status of the requested
                   operation when it has been completed.  Six status conditions are
                   possible on return:

                     0    The specified operation was completed without error.
                     1    The fdc is busy and the requested operation cannot be started.
                     2    Fdc error (further information is contained in the result
                          storage portion of the disk operation control block - as
                          described in the 8272 data sheet).
                     3    Transfer error during output of the command bytes to the fdc.
                     4    Transfer error during input of the result bytes from the fdc.
                     5    Invalid fdc command.    ****/

124   1      execute$docb: procedure(docb$ptr,status$ptr) public;
             /* execute a disk operation control block */

125   2        declare docb$ptr pointer, status$ptr pointer;
126   2        declare
                 docb based docb$ptr structure docb$type,
                 status based status$ptr byte,
                 drive$no byte;

               /* check command validity */
127   2        if not valid$command(command$code)
                 then do; status=stat$invalid; return; end;

               /* determine if command has a drive number field - if not, set the drive
                  number for a general fdc command */
132   2        if drive$no$present(command$code)
                 then drive$no=extract$drive$no;
134   2          else drive$no=fdc$general;

               /* an overlapped operation can not be performed if the fdc is busy */
135   2        if overlap$operation(command$code) and fdc$busy
                 then do; status=stat$busy; return; end;

               /* for a non-overlapped operation, check fdc busy or any drive seeking */
140   2        if not overlap$operation(command$code) and (fdc$busy or any$drive$seeking)
                 then do; status=stat$busy; return; end;

               /* check for drive operation in progress - if none, set flag and start operation */
145   2        disable;
146   2        if operation$in$progress(drive$no)
                 then do; enable; status=stat$busy; return; end;
152   2          else operation$in$progress(drive$no)=true;

               /* at this point, an fdc operation is about to begin, so:
                  1. reset the operation complete flag
                  2. set the docb pointer for the current operation
                  3. if this is not an overlapped operation, set the global drive
                     number for the subsequent result phase interrupt.  */
153   2        operation$complete(drive$no)=0;
154   2        operation$docb$ptr(drive$no)=docb$ptr;

155   2        if not overlap$operation(command$code)
                 then global$drive$no=drive$no+1;
157   2        enable;

158   2        call output$controls$to$dma(docb$ptr);
159   2        if error$in output$command$to$fdc(docb$ptr)
                 then do;
161   3            call operation$clean$up(drive$no,command$code);
162   3            status=stat$command$error;
163   3            return;
164   3          end;

               /* return immediately if the command has no result phase or completion interrupt - specify */
165   2        if no$result(command$code)
                 then do;
167   3            call operation$clean$up(drive$no,command$code);
168   3            status=stat$ok;
169   3            return;
170   3          end;
```

```
171  2        if immed$result(command$code)
                then do;
173  3           if error$in input$result$from$fdc(docb$ptr)
                   then do;
175  4              call operation$clean$up(drive$no,command$code);
176  4              status=stat$result$error;
177  4              return;
178  4           end;
179  3        end;
180  2        else do;
181  3           wait$for$op$complete;
183  3           if docb.misc = error
                   then do; status=stat$result$error; return; end;
188  3        end;

189  2     if no$fdc$error
              then status=stat$ok;
191  2        else status=stat$error;

192  2  end execute$docb;

        $eject

        /**** copy disk command results from the interrupt control block to the
              currently active disk operation control block if a disk operation is
              in progress.    ****/

193  1  copy$int$result: procedure(drv);
194  2     declare drv byte;
195  2     declare
              i byte,
              docb$ptr pointer,
              docb based docb$ptr structure docb$type;

196  2     if operation$in$progress(drv)
              then do;
198  3        docb$ptr=operation$docb$ptr(drv);
199  3        do i=1 to 6; docb.disk$result(i)=interrupt$docb.disk$result(i); end;
202  3        docb.misc=ok;
203  3        operation$in$progress(drv)=false;
204  3        operation$complete(drv)=true;
205  3        end;

206  2  end copy$int$result;


        /**** interrupt processing for 8272 fdc drivers.  Basically, two types of
              interrupts are generated by the 8272: (a)when the execution phase of
              an operation has been completed, an interrupt is generated to signal
              the beginning of the result phase (the fdc busy flag is set
              when this interrupt is received), and (b) when an overlapped operation
              is completed or an unexpected interrupt is received (the fdc busy flag
              is not set when this interrupt is received).

              When interrupt type (a) is received, the result bytes from the operation
              are read from the 8272 and the operation complete flag is set.

              When an interrupt of type (b) is received, the interrupt result code is
              examined to determine which of the following four actions are indicated:

                1. An overlapped option (recalibrate or seek) has been completed.  The
                   result data is read from the 8272 and placed in the currently active
                   disk operation control block.
                2. An abnormal termination of an operation has occurred.  The result
                   data is read and placed in the currently active disk operation
                   control block.
                3. The execution of an invalid command has been attempted.  This
                   signals the successful completion of all interrupt processing.
                4. The ready status of a drive has changed.  The "drive$ready" and
                   "drive$ready$status" change tables are updated.  If an operation
                   is currently in progress on the affected drive, the result data
                   is placed in the currently active disk operation control block.

              After an interrupt is processed, additional sense interrupt status commands
              must be issued and processed until an invalid command result is returned
              from the fdc.  This action guarantees that all "hidden" interrupts
              are serviced.    ****/
```

```
207   1     fdcint: procedure public interrupt fdc$int$level;
208   2       declare
                invalid byte,
                drive$no byte,
                docb$ptr pointer,
                docb based docb$ptr structure docb$type;

209   2       declare
                /* interrupt port definitions */
                ocw2                    literally '70H',
                nseoi                   literally 'shl(1,5)';

210   2       declare
                /* miscellaneous flags */
                result$code             literally 'shr(interrupt$docb.disk$result(0) and result$error$mask,6)',
                result$drive$ready      literally '((interrupt$docb.disk$result(0) and result$ready$mask) = 0)',
                extract$result$drive$no literally '(interrupt$docb.disk$result(0) and result$drive$mask)',
                end$of$interrupt        literally 'output(ocw2)=nseoi';

              /* if the fdc is busy when an interrupt is received, then the result
                 phase of the previous non-overlapped operation has begun */
211   2       if fdc$busy
                then do;
                  /* process interrupt if operation in progress */
213   3         if global$drive$no <> 0
                  then do;
215   4           docb$ptr=operation$docb$ptr(global$drive$no-1);
216   4             if error$in input$result$from$fdc(docb$ptr)
                      then docb.misc=error;
218   4                 else docb.misc=ok;
219   4             operation$in$progress(global$drive$no-1)=false;
220   4             operation$complete(global$drive$no-1)=true;
221   4             global$drive$no=0;
222   4           end;
223   3         end;

              /* if the fdc is not busy, then either an overlapped operation has been
                 completed or an unexpected interrupt has occurred (e.g., drive status
                 change) */
224   2         else do;
225   3           invalid=false;
226   3           do while not invalid;

                  /* perform a sense interrupt status operation - if errors are detected,
                     in the actual fdc interface, interrupt processing is discontinued */
227   4           if error$in output$byte$to$fdc(sense$int$status) then go to ignore;
229   4           if error$in input$result$from$fdc(@interrupt$docb) then go to ignore;

231   4           do case result$code;

                    /* case 0 - operation complete */
232   5             do;
233   6               drive$no=extract$result$drive$no;
234   6               call copy$int$result(drive$no);
235   6             end;

                    /* case 1 - abnormal termination */

236   5             do;
237   6               drive$no=extract$result$drive$no;
238   6               call copy$int$result(drive$no);
239   6             end;

                    /* case 2 - invalid command */
240   5             invalid=true;

                    /* case 3 - drive ready change */
241   5             do;
242   6               drive$no=extract$result$drive$no;
243   6               call copy$int$result(drive$no);
244   6               drive$status$change(drive$no)=true;
245   6               if result$drive$ready
                        then drive$ready(drive$no)=true;
247   6                 else drive$ready(drive$no)=false;
248   6             end;
249   5           end;
250   4         end;
251   3       end;

252   2     ignore: end$of$interrupt;
253   2   end fdcint;

254   1   end drivers;
```

```
MODULE INFORMATION:
     CODE AREA SIZE     = 0615H   1557D
     CONSTANT AREA SIZE = 0000H      0D
     VARIABLE AREA SIZE = 0050H     80D
     MAXIMUM STACK SIZE = 0032H     50D
     564 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-86 COMPILATION
```

APPENDIX B
8272 FDC EXERCISER PROGRAM

```
PL/M-86 COMPILER    8272 FLOPPY DISK DRIVER EXERCISE PROGRAM

ISIS-II PL/M-86 V1.2 COMPILATION OF MODULE RUN72
OBJECT MODULE PLACED IN :F1:run72.OBJ
COMPILER INVOKED BY:  plm86 :F1:run72.p86 DEBUG

                $title ('8272 floppy disk driver exercise program')
                $nointvector
                $optimize(2)
                $large
  1             run72: do;

  2   1           declare
                    docb$type                 literally          /* disk operation control block */
                       (dma$op byte,dma$addr word,dma$addr$ext byte,dma$count word,
                       disk$command(9) byte,disk$result(7) byte,misc byte)';

  3   1           declare
                    /* 8272 fdc commands */
                    fm                        literally '0',
                    mfm                       literally '1',
                    dma$mode                  literally '0',
                    non$dma$mode              literally '1',
                    recalibrate$command       literally '7',
                    specify$command           literally '3',
                    read$command              literally '6',
                    write$command             literally '5',
                    format$command            literally '0DH',
                    seek$command              literally '0FH';

  4   1           declare
                    dma$verify                literally '0',
                    dma$read                  literally '1',
                    dma$write                 literally '2',
                    dma$noop                  literally '3';

  5   1           declare
                    /* disk operation control blocks */
                    format$docb               structure docb$type,
                    seek$docb                 structure docb$type,
                    recalibrate$docb          structure docb$type,
                    specify$docb              structure docb$type,
                    read$docb                 structure docb$type,
                    write$docb                structure docb$type;

  6   1           declare
                    step$rate                 byte,
                    head$load$time            byte,
                    head$unload$time          byte,
                    filler$byte               byte,
                    operation$status          byte,
                    interleave                byte,
                    format$gap                byte,
                    read$write$gap            byte,
                    index                     byte,
                    drive                     byte,
                    density                   byte,
                    multitrack                byte,
                    sector                    byte,
                    cylinder                  byte,
                    head                      byte,              /* disk drive head */
                    tracks$per$disk           byte,
                    sectors$per$track         byte,
                    bytes$per$sector$code     byte,
                    bytes$per$sector          word;              /* number of bytes in a sector on the disk */

  7   1           declare
                    /* read and write buffers */
                    fmtblk(104)               byte public,
                    wrbuf(1024)               byte public,
                    rdbuf(1024)               byte public;

  8   1           declare
                    /* disk format initialization tables */
                    sec$trk$table(3)          byte data(26,15,8),
                    fmt$gap$table(8)          byte data(1BH,2AH,3AH,0,0,36H,54H,74H),
                    rd$wr$gap$table(8)        byte data(07H,0EH,1BH,0,0,0EH,1BH,35H);
```

```
 9   1        declare
                /* external pointer tables and interrupt vector */
                rdbptr(2)                word external,
                wrbptr(2)                word external,
                fbptr(2)                 word external,
                intptr(2)                word external,
                intvec(80H)              word external;


10   1        execute$docb: procedure(docb$ptr,status$ptr) external;
11   2          declare docb$ptr pointer, status$ptr pointer;
12   2        end execute$docb;

13   1        initialize$drivers: procedure external;
14   2        end initialize$drivers;

            $eject

            /**** specify step rate ("srt"), head load time ("hlt"), head unload time ("hut"),
                  and dma or non-dma operation ("nd").   ****/

15   1        specify: procedure(srt,hlt,hut,nd);
16   2          declare (srt,hlt,hut,nd) byte;

17   2          specify$docb.dma$op=dma$noop;
18   2          specify$docb.disk$command(0)=specify$command;
19   2          specify$docb.disk$command(1)=shl((not srt)+1,4) or shr(hut,4);
20   2          specify$docb.disk$command(2)=(hlt and 0FEH) or (nd and 1);
21   2          call execute$docb(@specify$docb,@operation$status);

22   2        end specify;


            /****  recalibrate disk drive
                  8272 automatically steps out until the track 0 signal is activated
                  by the disk drive.   ****/

23   1        recalibrate: procedure(drv);
24   2          declare drv byte;

25   2          recalibrate$docb.dma$op=dma$noop;
26   2          recalibrate$docb.disk$command(0)=recalibrate$command;
27   2          recalibrate$docb.disk$command(1)=drv;
28   2          call execute$docb(@recalibrate$docb,@operation$status);

29   2        end recalibrate;


            /**** seek drive "drv", head (side) "hd" to cylinder "cyl".   ****/

30   1        seek: procedure(drv,cyl,hd);
31   2          declare (drv,cyl,hd) byte;

32   2          seek$docb.dma$op=dma$noop;
33   2          seek$docb.disk$command(0)=seek$command;
34   2          seek$docb.disk$command(1)=drv or shl(hd,2);
35   2          seek$docb.disk$command(2)=cyl;
36   2          call execute$docb(@seek$docb,@operation$status);

37   2        end seek;


            /**** format a complete side ("head") of a single floppy disk in drive "drv".  The density,
                  (single or double) is specified by flag "dens".   ****/

38   1        format: procedure(drv,dens,intlve);
               /*  format disk */
39   2          declare (drv,dens,intlve) byte;
40   2          declare physical$sector byte;

41   2          call recalibrate(drv);
42   2          do cylinder=0 to tracks$per$disk-1;
                  /* set sector numbers in format block to zero before computing interleave */

43   3            do physical$sector=1 to sectors$per$track; fmtblk((physical$sector-1)*4+2)=0; end;
                  /* physical sector 1 equals logical sector 1 */
46   3            physical$sector=1;

                  /* assign interleaved sectors */
47   3            do sector=1 to sectors$per$track;
48   4              index=(physical$sector-1)*4;
```

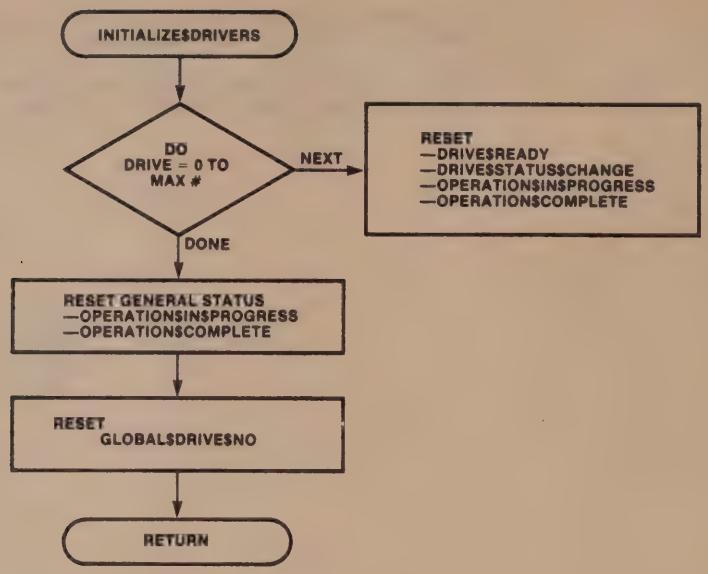
```
                /* change sector and index if sector has already been assigned */
 49   4         do while fmtblk(index+2) <> 0; index=index+4; physical$sector=physical$sector+1; end;

                /* set cylinder, head, sector, and size code for current sector into table */
 53   4         fmtblk(index)=cylinder;
 54   4         fmtblk(index+1)=head;
 55   4         fmtblk(index+2)=sector;
 56   4         fmtblk(index+3)=bytes$per$sector$code;

                /* update physical sector number by interleave */
 57   4         physical$sector=physical$sector+intlve;
 58   4         if physical$sector > sectors$per$track
                  then physical$sector=physical$sector-sectors$per$track;
 60   4       end;

              /* seek to next cylinder */
 61   3       call seek(drv,cylinder,head);

              /* set up format control block */
 62   3       format$docb.dma$op=dma$write;
 63   3       format$docb.dma$addr=fbptr(0)+shl(fbptr(1),4);
 64   3       format$docb.dma$addr$ext=0;
 65   3       format$docb.dma$count=sectors$per$track*4-1;
 66   3       format$docb.disk$command(0)=format$command or shl(dens,6);
 67   3       format$docb.disk$command(1)=drv or shl(head,2);
 68   3       format$docb.disk$command(2)=bytes$per$sector$code;
 69   3       format$docb.disk$command(3)=sectors$per$track;
 70   3       format$docb.disk$command(4)=format$gap;
 71   3       format$docb.disk$command(5)=filler$byte;
 72   3       call execute$docb(@format$docb,@operation$status);
 73   3     end;

 74   2   end format;


        /**** write sector "sec" on drive "drv" at head "hd" and cylinder "cyl".  The
              disk recording density is specified by the "dens" flag.  Data is expected to be
              in the global write buffer ("wrbuf").    ****/

 75   1   write: procedure(drv,cyl,hd,sec,dens);
 76   2     declare (drv,cyl,hd,sec,dens) byte;

 77   2     write$docb.dma$op=dma$write;
 78   2     write$docb.dma$addr=wrbptr(0)+shl(wrbptr(1),4);
 79   2     write$docb.dma$addr$ext=0;
 80   2     write$docb.dma$count=bytes$per$sector-1;
 81   2     write$docb.disk$command(0)=write$command or shl(dens,6) or shl(multitrack,7);
 82   2     write$docb.disk$command(1)=drv or shl(hd,2);
 83   2     write$docb.disk$command(2)=cyl;
 84   2     write$docb.disk$command(3)=hd;
 85   2     write$docb.disk$command(4)=sec;
 86   2     write$docb.disk$command(5)=bytes$per$sector$code;
 87   2     write$docb.disk$command(6)=sectors$per$track;
 88   2     write$docb.disk$command(7)=read$write$gap;
 89   2     if bytes$per$sector$code = 0
              then write$docb.disk$command(8)=bytes$per$sector;
 91   2       else write$docb.disk$command(8)=0FFH;
 92   2     call execute$docb(@write$docb,@operation$status);

 93   2   end write;


        /**** read sector "sec" on drive "drv" at head "hd" and cylinder "cyl".  The
              disk recording density is defined by the "dens" flag.  Data is read into
              the global read buffer ("rdbuf").    ****/

 94   1   read: procedure(drv,cyl,hd,sec,dens);
 95   2     declare (drv,cyl,hd,sec,dens) byte;

 96   2     read$docb.dma$op=dma$read;
 97   2     read$docb.dma$addr=rdbptr(0)+shl(rdbptr(1),4);
 98   2     read$docb.dma$addr$ext=0;
 99   2     read$docb.dma$count=bytes$per$sector-1;
100   2     read$docb.disk$command(0)=read$command or shl(dens,6) or shl(multitrack,7);
101   2     read$docb.disk$command(1)=drv or shl(hd,2);
102   2     read$docb.disk$command(2)=cyl;
103   2     read$docb.disk$command(3)=hd;
104   2     read$docb.disk$command(4)=sec;
105   2     read$docb.disk$command(5)=bytes$per$sector$code;
106   2     read$docb.disk$command(6)=sectors$per$track;
107   2     read$docb.disk$command(7)=read$write$gap;
```

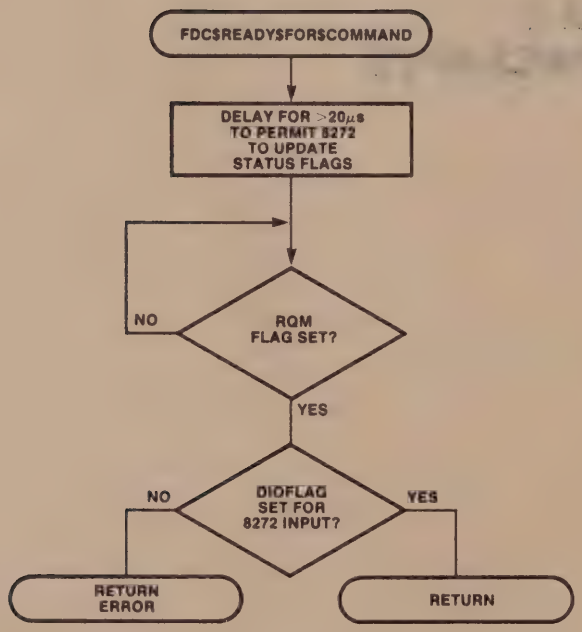
```
108  2        if bytes$per$sector$code = 0
                then read$docb.disk$command(8)=bytes$per$sector;
110  2          else read$docb.disk$command(8)=0FFH;
111  2        call execute$docb(@read$docb,@operation$status);

112  2      end read;

            $eject

            /**** initialize system by setting up 8237 dma controller and 8259A interrupt
                  controller.    ****/

113  1      initialize$system: procedure;
114  2        declare
                /* I/O ports */
                dma$disk$addr$port        literally '00H',      /* current address port */
                dma$disk$word$count$port literally '01H',       /* word count port */
                dma$command$port          literally '08H',      /* command port */
                dma$mode$port             literally '0BH',      /* mode port */
                dma$mask$sr$port          literally '0AH',      /* mask set/reset port */
                dma$clear$ff$port         literally '0CH',      /* clear first/last flip-flop port */
                dma$master$clear$port     literally '0DH',      /* dma master clear port */
                dma$mask$port             literally '0FH',      /* parallel mask set port*/
                dma$c1$addr$port          literally '02H',
                dma$c1$word$count$port    literally '03H',
                dma$c2$addr$port          literally '04H',
                dma$c2$word$count$port    literally '05H',
                dma$c3$addr$port          literally '06H',
                dma$c3$word$count$port    literally '07H',
                icw1                      literally '70H',
                icw2                      literally '71H',
                icw4                      literally '71H',
                ocw1                      literally '71H',
                ocw2                      literally '70H',
                ocw3                      literally '70H';

115  2        declare
                /* misc masks and literals */
                dma$extended$write        literally 'shl(1,5)',   /* extended write flag */
                dma$single$transfer       literally 'shl(1,6)',   /* single transfer flag */
                dma$disk$mode             literally '40H',
                dma$c1$mode               literally '41H',
                dma$c2$mode               literally '42H',
                dma$c3$mode               literally '43H',
                mode$8088                 literally '1',
                interrupt$base            literally '20H',
                single$controller         literally 'shl(1,1)',
                level$sensitive           literally 'shl(1,3)',
                control$word$4$required   literally '1',
                base$icw1                 literally '10H',
                mask$all                  literally '0FFH',
                disk$interrupt$mask       literally '1';

116  2        output(dma$master$clear$port)=0;          /* master reset */
117  2        output(dma$mode$port)=dma$extended$write;  /* set dma command mode */

              /* set all dma registers to valid values */
118  2        output(dma$mask$port)=mask$all;            /* mask all channels */

              /* set all addresses to zero */
119  2        output(dma$clear$ff$port)=0;               /* reset first/last flip-flop */
120  2        output(dma$disk$addr$port)=0;
121  2        output(dma$disk$addr$port)=0;
122  2        output(dma$c1$addr$port)=0;
123  2        output(dma$c1$addr$port)=0;
124  2        output(dma$c2$addr$port)=0;
125  2        output(dma$c2$addr$port)=0;
126  2        output(dma$c3$addr$port)=0;
127  2        output(dma$c3$addr$port)=0;

              /* set all word counts to valid values */
128  2        output(dma$clear$ff$port)=0;               /* reset first/last flip-flop */
129  2        output(dma$disk$word$count$port)=1;
130  2        output(dma$disk$word$count$port)=1;
131  2        output(dma$c1$word$count$port)=1;
132  2        output(dma$c1$word$count$port)=1;
133  2        output(dma$c2$word$count$port)=1;
134  2        output(dma$c2$word$count$port)=1;
135  2        output(dma$c3$word$count$port)=1;
136  2        output(dma$c3$word$count$port)=1;
```

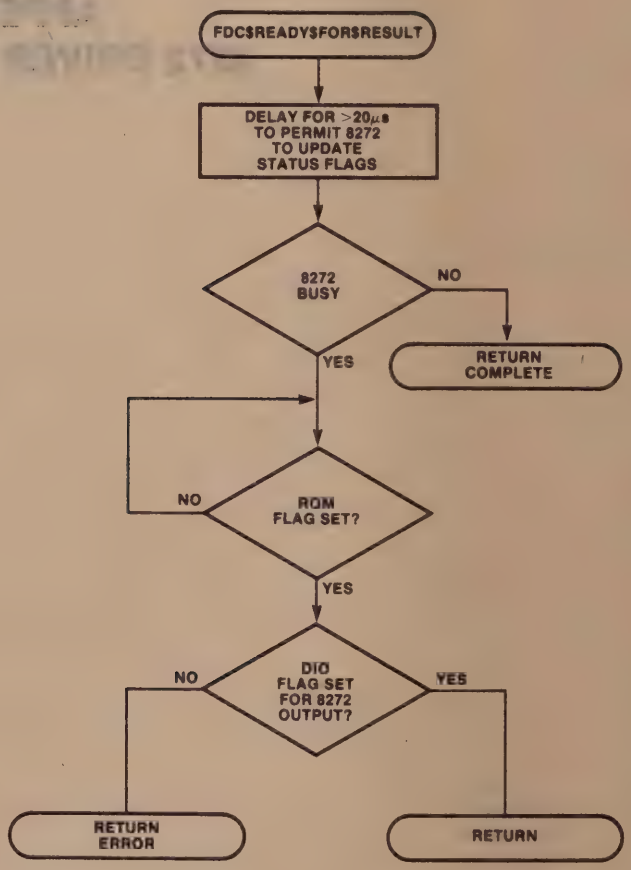
```
                /* initialize all dma channel modes */
137   2         output(dma$mode$port)=dma$disk$mode;
138   2         output(dma$mode$port)=dma$c1$mode;
139   2         output(dma$mode$port)=dma$c2$mode;
140   2         output(dma$mode$port)=dma$c3$mode;

                /* initialize 8259A interrupt controller */
141   2         output(icw1)=single$controller or level$sensitive or control$word4$required or base$icw1;
142   2         output(icw2)=interrupt$base;
143   2         output(icw4)=mode$8088;                        /* set 8088 interrupt mode */
144   2         output(ocw1)=not disk$interrupt$mask;          /* mask all interrupts except disk */

                /* initialize interrupt vector for fdc */
145   2         intvec(40H)=intptr(0);
146   2         intvec(41H)=intptr(1);

147   2       end initialize$system;

              $eject

              /**** main program:   first format disk (all tracks on side (head) 0.  Then
                    read each sector on every track of the disk forever.    ****/

148   1       declare drive$ready(4) byte external;

                /* disable until interrupt vector setup and initialization complete */
149   1       disable;

                /* set initial floppy disk parameters */
150   1       density=mfm;                                   /* double-density */
151   1       head=0;                                        /* single sided */
152   1       multitrack=0;                                  /* no multitrack operation */
153   1       filler$byte=55H;                               /* for format */
154   1       tracks$per$disk=77;                            /* normal floppy disk drive */
155   1       bytes$per$sector=1024;                         /* 1024 bytes in each sector */
156   1       interleave=6;                                  /* set track interleave factor */
157   1       step$rate=11;                                  /* 10ms for SA800 plus 1 for uncertainty */
158   1       head$load$time=40;                             /* 40ms head load for SA800 */
159   1       head$unload$time=240;                          /* keep head loaded as long as possible */

                /* derive dependent parameters from those above */
160   1       bytes$per$sector$code=shr(bytes$per$sector,7);
161   1       do index=0 to 3;
162   2         if (bytes$per$sector$code and 1) <> 0
                    then do; bytes$per$sector$code=index; go to donebc; end;
167   2         else bytes$per$sector$code=shr(bytes$per$sector$code,1);
168   2       end;

169   1       donebc:
              sectors$per$track=sec$trk$table(bytes$per$sector$code-density);
170   1       format$gap=fmt$gap$table(shl(density,2)+bytes$per$sector$code);
171   1       read$write$gap=rd$wr$gap$table(shl(density,2)+bytes$per$sector$code);

                /* initialize system and drivers */
172   1       call initialize$system;
173   1       call initialize$drivers;

                /* reenable interrupts and give 8272 a chance to report on drive status
                   before proceeding */
174   1       enable;
175   1       call time(10);

                /* specify disk drive parameters */
176   1       call specify(step$rate,head$load$time,head$unload$time,dma$mode);

177   1       drive=0;                                       /* run single disk drive #0 */

                /* wait until drive ready */
178   1       do while 1;
179   2         if drive$ready(drive)
                    then go to start;
181   2       end;

182   1       start:
              call format(drive,density,interleave);

183   1       do while 1;
184   2         do cylinder=0 to tracks$per$disk-1;
185   3           call seek(drive,cylinder,head);
186   3           do sector=1 to sectors$per$track;

                    /* set up write buffer */
187   4             do index=0 to bytes$per$sector-1; wrbuf(index)=index+sector+cylinder; end;
```

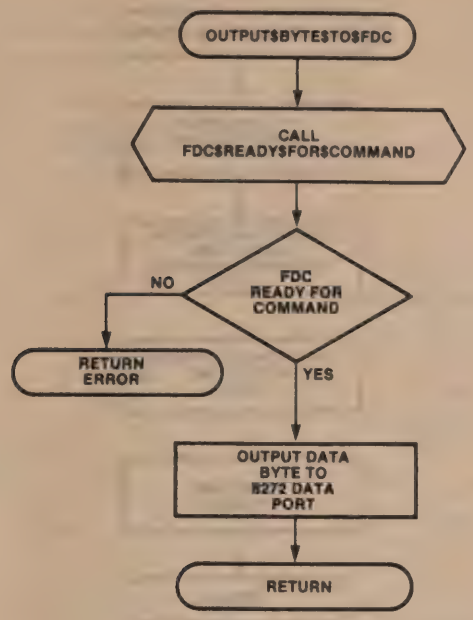
```
 190   4              call write(drive,cylinder,head,sector,density);
 191   4              call read(drive,cylinder,head,sector,density);

                      /* check read buffer against write buffer */
 192   4              if cmpw(@wrbuf,@rdbuf,shr(bytes$per$sector,1)) <> 0FFFFH
                        then halt;
 194   4            end;
 195   3          end;
 196   2        end;

 197   1      end run72;

MODULE INFORMATION:

     CODE AREA SIZE     = 0570H   1392D
     CONSTANT AREA SIZE = 0000H      0D
     VARIABLE AREA SIZE = 0907H   2311D
     MAXIMUM STACK SIZE = 0022H     34D
     412 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-86 COMPILATION
```

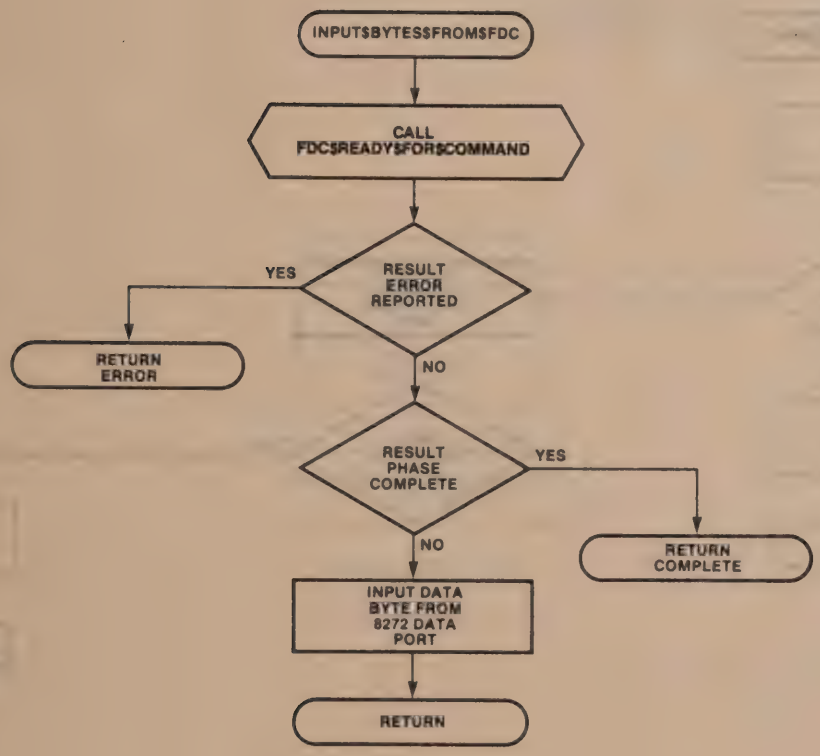
APPENDIX C
8272 DRIVER FLOWCHARTS

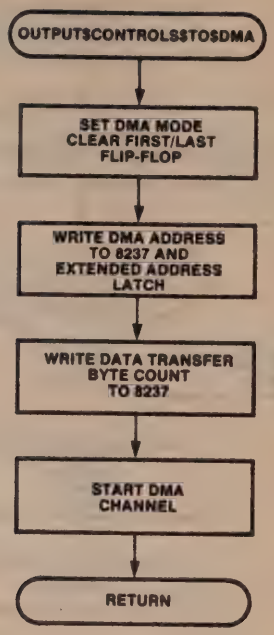

INITIALIZE$DRIVERS
DO
DRIVE = 0 TO
MAX #
NEXT
RESET
—DRIVE$READY
—DRIVE$STATUS$CHANGE
—OPERATIONINPROGRESS
—OPERATION$COMPLETE
DONE
RESET GENERAL STATUS
—OPERATIONINPROGRESS
—OPERATION$COMPLETE
RESET
GLOBAL$DRIVE$NO
RETURN

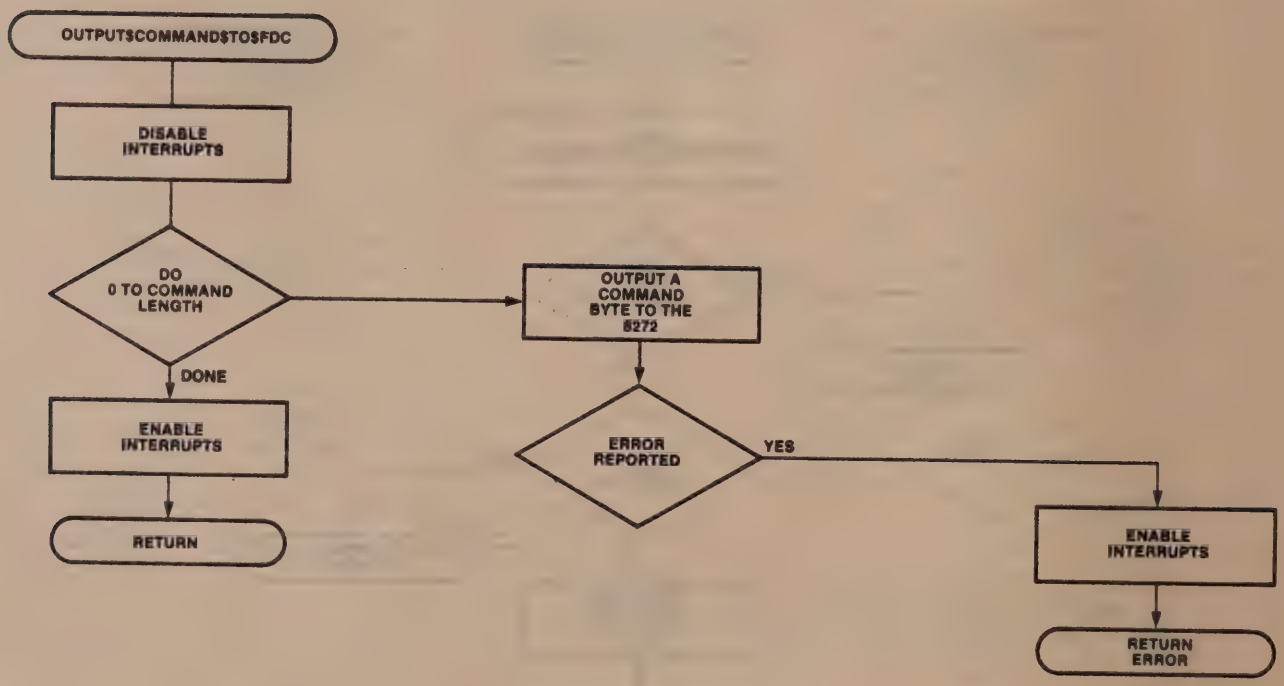

FDC$READY$FOR$COMMAND
DELAY FOR >20μs
TO PERMIT 8272
TO UPDATE
STATUS FLAGS
NO
RQM
FLAG SET?
YES
NO
DIOFLAG
SET FOR
8272 INPUT?
YES
RETURN
ERROR
RETURN

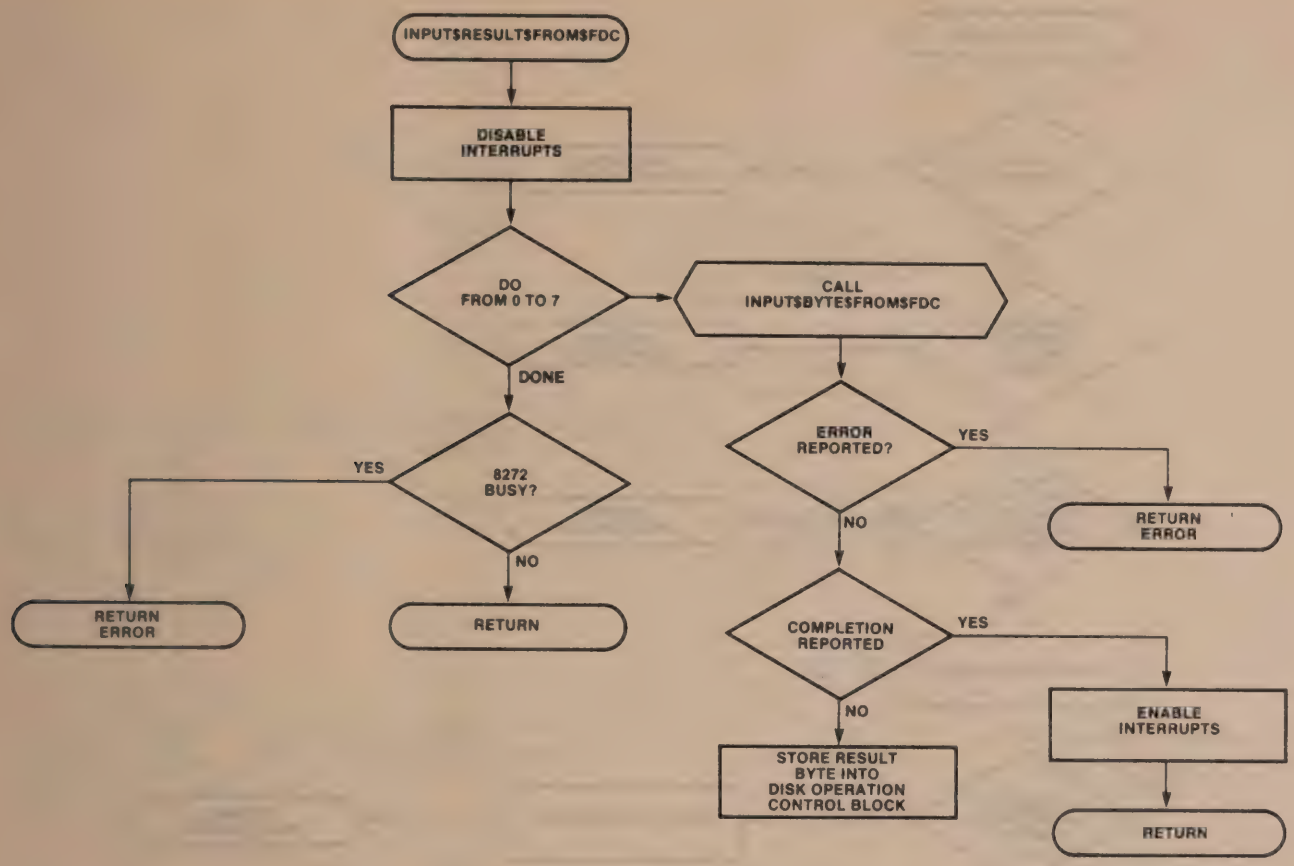

FDC$READY$FOR$RESULT
DELAY FOR >20μs
TO PERMIT 8272
TO UPDATE
STATUS FLAGS
8272
BUSY
NO
RETURN
COMPLETE
YES
NO
RQM
FLAG SET?
YES
NO
DIO
FLAG SET
FOR 8272
OUTPUT?
YES
RETURN
ERROR
RETURN

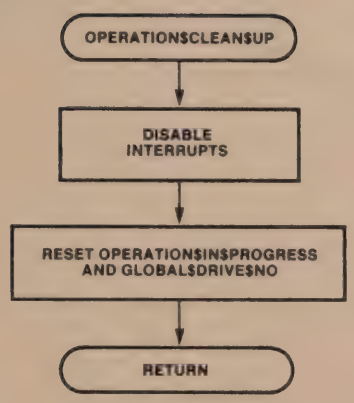

OUTPUT$BYTESTO$FDC
CALL
FDC$READY$FOR$COMMAND
FDC
READY FOR
COMMAND
NO
RETURN
ERROR
YES
OUTPUT DATA
BYTE TO
8272 DATA
PORT
RETURN

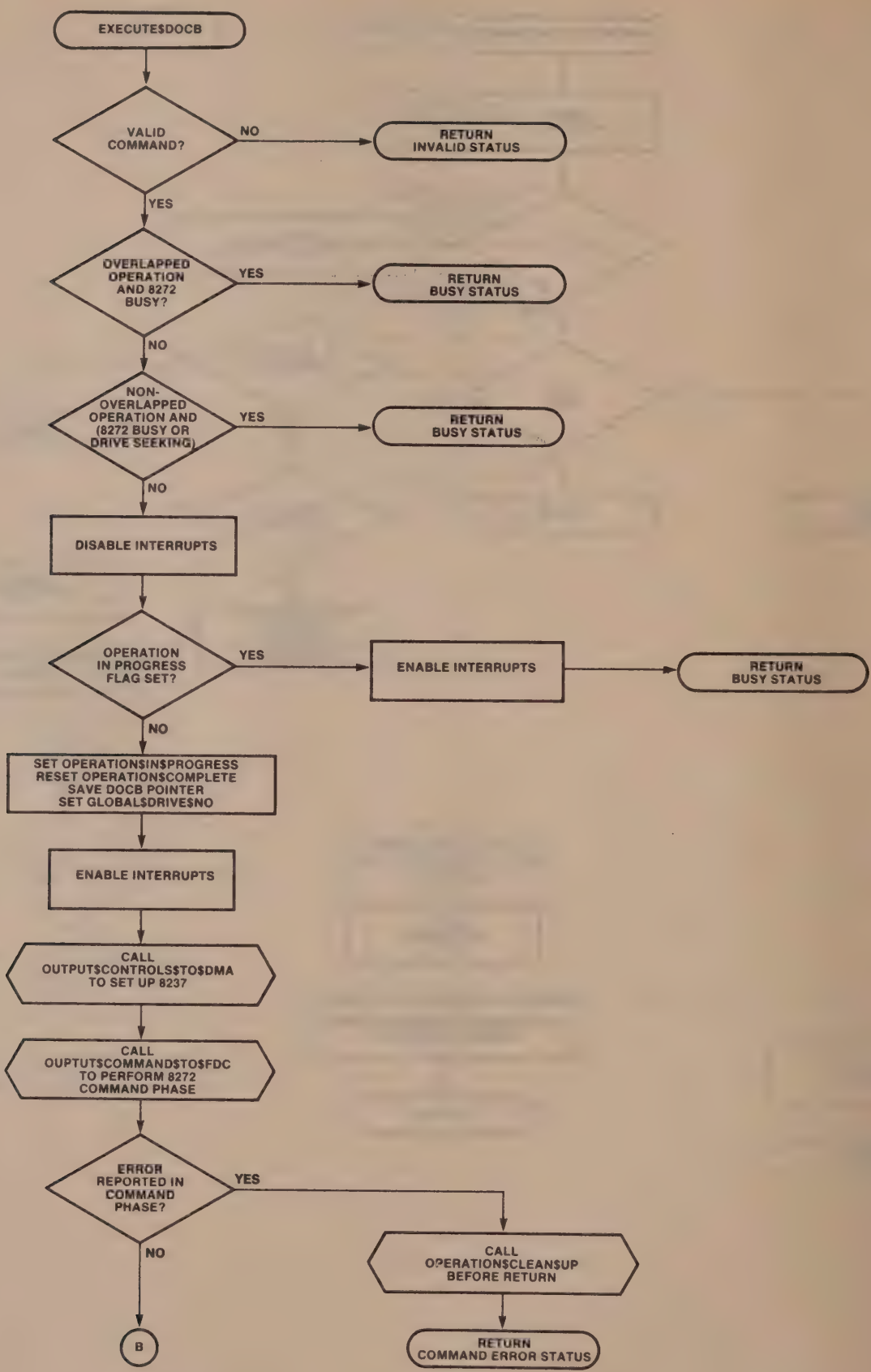

INPUT$BYTESSFROM$FDC
CALL
FDC$READY$FOR$COMMAND
RESULT
ERROR
REPORTED
YES
RETURN
ERROR
NO
RESULT
PHASE
COMPLETE
YES
RETURN
COMPLETE
NO
INPUT DATA
BYTE FROM
8272 DATA
PORT
RETURN

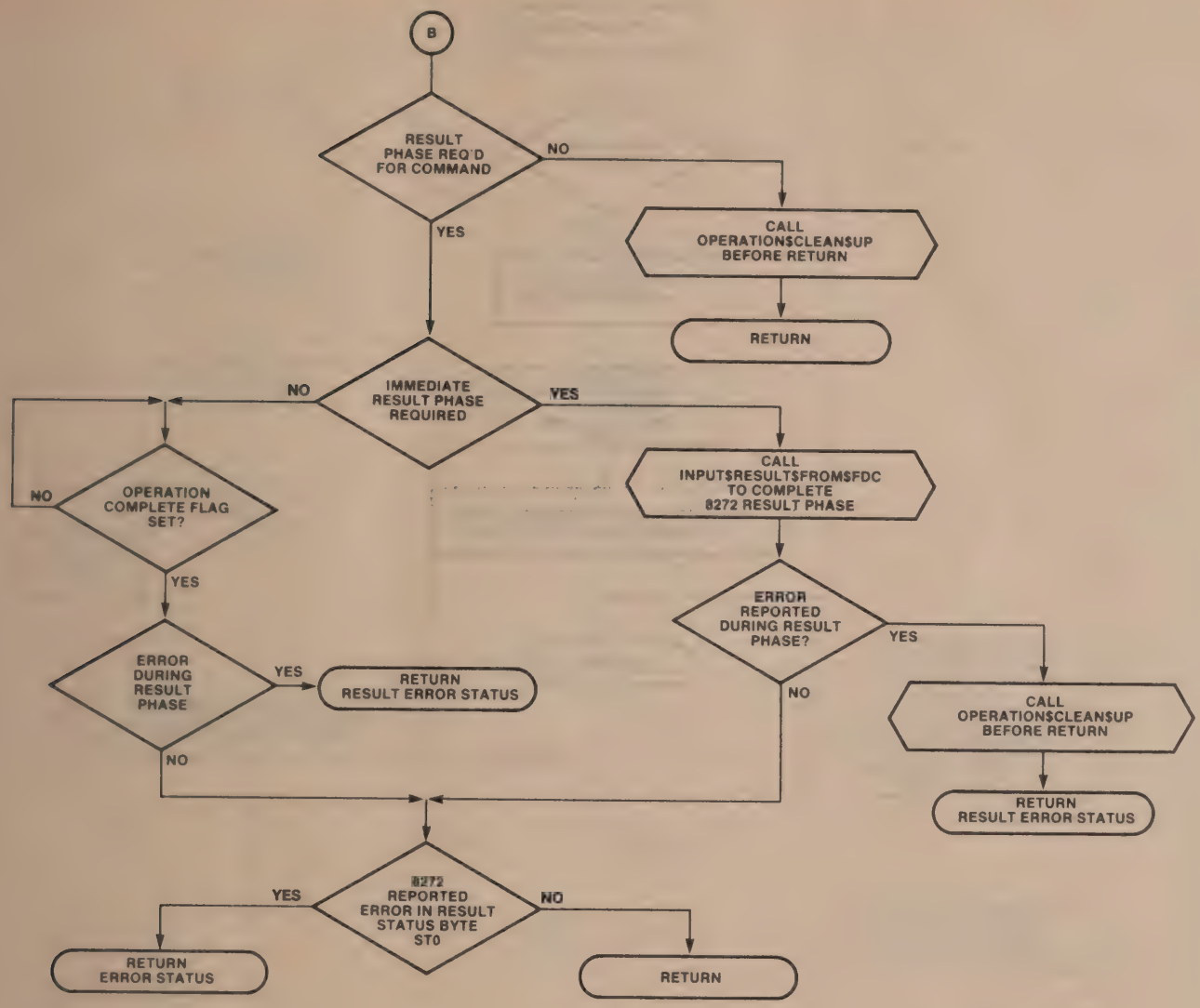

OUTPUT$CONTROLS$TO$DMA
SET DMA MODE
CLEAR FIRST/LAST
FLIP-FLOP
WRITE DMA ADDRESS
TO 8237 AND
EXTENDED ADDRESS
LATCH
WRITE DATA TRANSFER
BYTE COUNT
TO 8237
START DMA
CHANNEL
RETURN

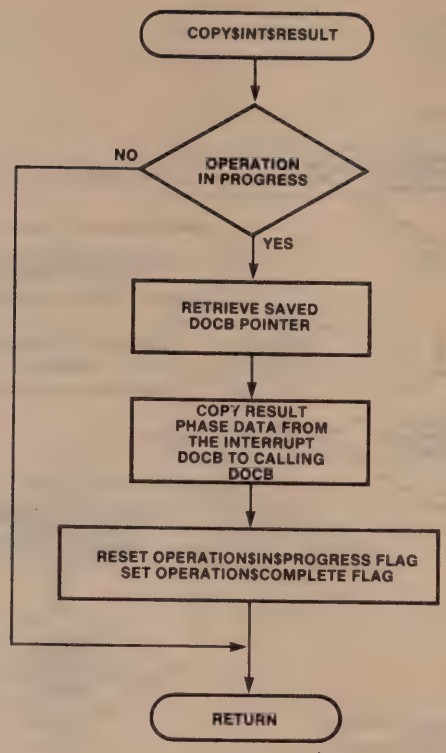

OUTPUT$COMMAND$TO$FDC
DISABLE
INTERRUPTS
DO
0 TO COMMAND
LENGTH
OUTPUT A
COMMAND
BYTE TO THE
8272
DONE
ENABLE
INTERRUPTS
RETURN
ERROR
REPORTED
YES
ENABLE
INTERRUPTS
RETURN
ERROR

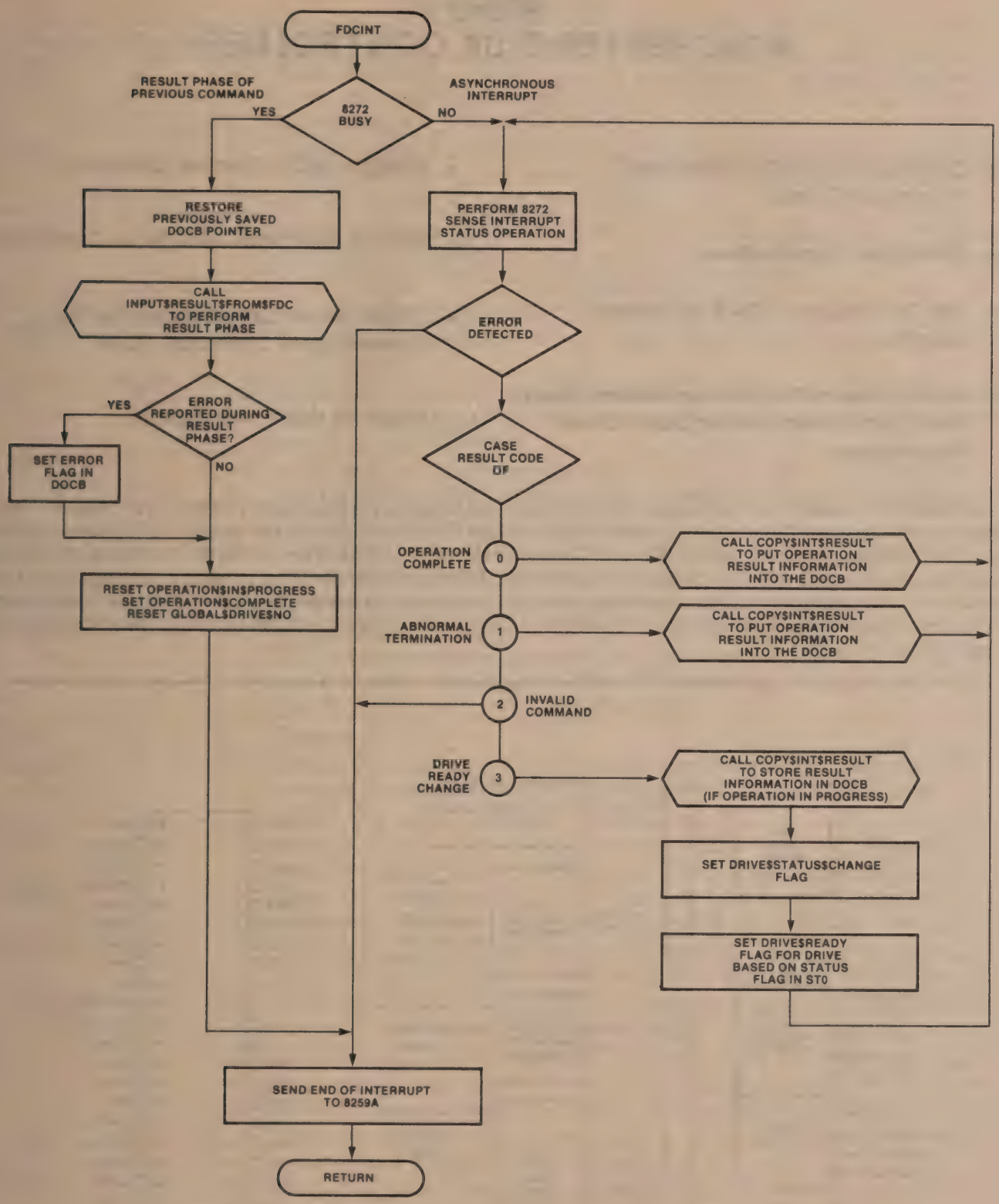

INPUT$RESULT$FROM$FDC
DISABLE INTERRUPTS
DO FROM 0 TO 7
CALL INPUT$BYTES$FROM$FDC
DONE
8272 BUSY?
YES
NO
RETURN ERROR
RETURN
ERROR REPORTED?
YES
NO
RETURN ERROR
COMPLETION REPORTED
YES
NO
STORE RESULT BYTE INTO DISK OPERATION CONTROL BLOCK
ENABLE INTERRUPTS
RETURN


OPERATION$CLEAN$UP
DISABLE INTERRUPTS
RESET OPERATIONINPROGRESS AND GLOBAL$DRIVE$NO
RETURN


EXECUTE$DOCB
VALID COMMAND?
NO
RETURN INVALID STATUS
YES
OVERLAPPED OPERATION AND 8272 BUSY?
YES
RETURN BUSY STATUS
NO
NON-OVERLAPPED OPERATION AND (8272 BUSY OR DRIVE SEEKING)
YES
RETURN BUSY STATUS
NO
DISABLE INTERRUPTS
OPERATION IN PROGRESS FLAG SET?
YES
ENABLE INTERRUPTS
RETURN BUSY STATUS
NO
SET OPERATIONINPROGRESS
RESET OPERATION$COMPLETE
SAVE DOCB POINTER
SET GLOBAL$DRIVE$NO
ENABLE INTERRUPTS
CALL OUTPUT$CONTROLS$TO$DMA TO SET UP 8237
CALL OUPTUT$COMMAND$TO$FDC TO PERFORM 8272 COMMAND PHASE
ERROR REPORTED IN COMMAND PHASE?
YES
NO
CALL OPERATION$CLEAN$UP BEFORE RETURN
RETURN COMMAND ERROR STATUS
B


B
RESULT PHASE REQ'D FOR COMMAND
NO
YES
CALL OPERATION$CLEAN$UP BEFORE RETURN
RETURN
IMMEDIATE RESULT PHASE REQUIRED
NO
YES
OPERATION COMPLETE FLAG SET?
NO
YES
CALL INPUT$RESULT$FROM$FDC TO COMPLETE 8272 RESULT PHASE
ERROR DURING RESULT PHASE
YES
RETURN RESULT ERROR STATUS
NO
ERROR REPORTED DURING RESULT PHASE?
YES
NO
CALL OPERATION$CLEAN$UP BEFORE RETURN
RETURN RESULT ERROR STATUS
8272 REPORTED ERROR IN RESULT STATUS BYTE ST0
YES
NO
RETURN ERROR STATUS
RETURN


COPYINTRESULT
OPERATION IN PROGRESS
NO
YES
RETRIEVE SAVED DOCB POINTER
COPY RESULT PHASE DATA FROM THE INTERRUPT DOCB TO CALLING DOCB
RESET OPERATIONINPROGRESS FLAG
SET OPERATION$COMPLETE FLAG
RETURN


FDCINT
RESULT PHASE OF PREVIOUS COMMAND
YES
8272 BUSY
NO
ASYNCHRONOUS INTERRUPT
RESTORE PREVIOUSLY SAVED DOCB POINTER
PERFORM 8272 SENSE INTERRUPT STATUS OPERATION
CALL INPUT$RESULT$FROM$FDC TO PERFORM RESULT PHASE
ERROR DETECTED
YES
ERROR REPORTED DURING RESULT PHASE?
SET ERROR FLAG IN DOCB
NO
CASE RESULT CODE OF
OPERATION COMPLETE
0
CALL COPYINTRESULT TO PUT OPERATION RESULT INFORMATION INTO THE DOCB
RESET OPERATIONINPROGRESS
SET OPERATION$COMPLETE
RESET GLOBAL$DRIVE$NO
ABNORMAL TERMINATION
1
CALL COPYINTRESULT TO PUT OPERATION RESULT INFORMATION INTO THE DOCB
2
INVALID COMMAND
DRIVE READY CHANGE
3
CALL COPYINTRESULT TO STORE RESULT INFORMATION IN DOCB (IF OPERATION IN PROGRESS)
SET DRIVE$STATUS$CHANGE FLAG
SET DRIVE$READY FLAG FOR DRIVE BASED ON STATUS FLAG IN ST0
SEND END OF INTERRUPT TO 8259A
RETURN

82062
WINCHESTER DISK CONTROLLER

- Controls ST506/ST412 Interface Winchester Drives
- 5 MBit/Sec Transfer Rate
- 128, 256, 512, and 1024 Byte Sector Lengths
- Six High-Level Commands: Restore, Seek, Read Sector, Write Sector, Scan ID, and Write Format
- Multiple Sector Transfer Capability
- Implied Seek With Read/Write Commands
- 7 Byte Sector Length Extension For External Error Correction Code
- Single +5 Volt Power Supply

The 82062 Winchester Disk Controller (WDC) device interfaces microprocessor systems to Winchester Disks that use the Seagate Technology ST506/ST412 interface. Examples include the Seagate ST506 and ST412, Shugart SA604 and SA606, Tandon 600, and Computer Memories CM5206 and CM5412. The device translates parallel data from the microprocessor to a 5 mbit/sec, MFM-encoded serial bit stream. It provides all of the drive control logic and, in addition, control signals which simplify the design of an external phase locked loop and write precompensation circuitry. The 82062 is designed to interface to the host controller through an external sector buffer.

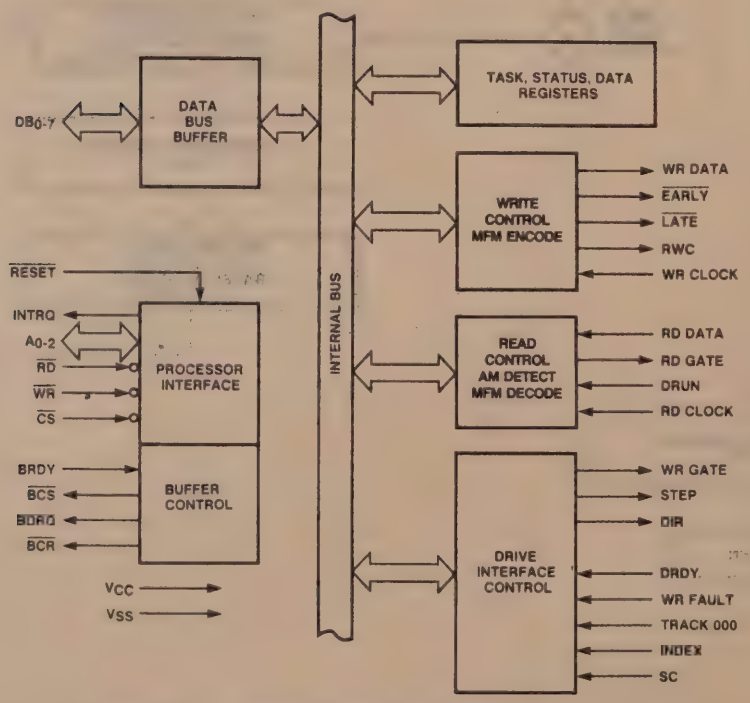


Figure 1. 82062 Block Diagram

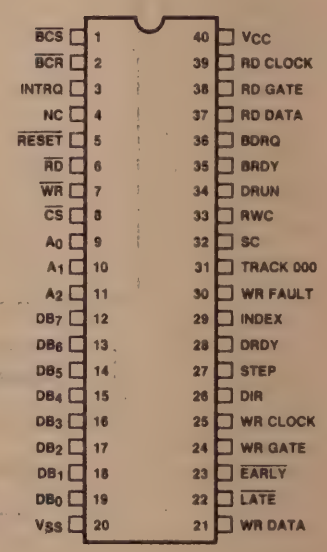


Figure 2. Pin Configuration

Table 1. Pin Description

Symbol	Pin No.	Type	Name and Function
$\overline{BCS}$	1	O	**Buffer Chip Select:** Output used to enable reading or writing of the external sector buffer by the 82062. When low, the host should not able to drive the 82062 data bus, $\overline{RD}$, or $\overline{WR}$ lines.
$\overline{BCR}$	2	O	**Buffer Counter Reset:** Output that is strobed by the 82062 prior to read/write operation. This pin is strobed whenever $\overline{BCS}$ changes state. Used to reset the address counter of the buffer memory.
INTRQ	3	O	**Interrupt Request:** Interrupt generated by the 82062 upon command termination. It is reset when any register is read. Optionally signifies when a data transfer is required on Read Sector commands.
N/C	4		No connection. Reserved for future use.
$\overline{RESET}$	5	I	**Reset:** Initializes the controller and clears all status flags. Does not clear the Task Registers.
$\overline{RD}$	6	I/O	**Read:** As an input, $\overline{RD}$ controls the transfer of information from the 82062 registers to the host. $\overline{RD}$ is an output when the 82062 is reading data from the sector buffer ($\overline{BCS}$ low).
$\overline{WR}$	7	I/O	**Write:** As an input, $\overline{WR}$ controls the transfer of command or task information into the 82062 registers. $\overline{WR}$ is an output when the 82062 is writing data to the sector buffer ($\overline{BCS}$ low).
$\overline{CS}$	8	I	**Chip Select:** Enables $\overline{RD}$ and $\overline{WR}$ as inputs for access to the Task Registers. It has no effect once a disk command starts..
A_0-A_2	9-11	I	**Address:** Used to select a register from the task register file.
DB_0-DB_7	12-19	I/O	**Data Bus:** Bidirectional 8-bit Data Bus with control determined by $\overline{BCS}$. When $\overline{BCS}$ is high the microprocessor has full control of the data bus for reading and writing the Task Registers. When $\overline{BCS}$ is low the 82062 controls the data bus to transfer data to or from the buffer.
GND	20		**Ground**
WR DATA	21	O	**Write Data:** Open drain output that shifts out MFM data at a rate determined by Write Clock. Requires an external flip-flop clocked at 10 MHz. See note 1.
$\overline{LATE}$	22	O	**Late:** Open drain output used to derive a delay value for write precompensation. Valid when WR GATE is high. Active on all cylinders. See note 1.
$\overline{EARLY}$	23	O	**Early:** Open drain output used to derive a delay value for write precompensation. Valid when WR GATE is high. Active on all cylinders. See note 1.
WR GATE	24	O	**Write Gate:** High when write data is valid. WR GATE goes low if the WR FAULT input is active. This output is used by the drive to enable head write current.
WR CLOCK	25	I	**Write Clock:** Clock input used to derive the write data rate. Frequency - 5MHz for the ST506 interface, 4.34MHz for the SA 1000 interface. See note 2.
DIR	26	O	**Direction:** High level on this output tells the drive to move the head inward (increasing cylinder number). The state of this signal is determined by the 82062's internal comparison of actual cylinder location vs desired cylinder.
STEP	27	O	**Step:** Provides 8.4 microsecond pulses to move the drive head to another cylinder at a programmable frequency.
DRDY	28	I	**Drive Ready:** If DRDY from the drive goes low, the command will be terminated.

Table 1. Pin Description (continued)

Symbol	Pin No.	Type	Name and Function
INDEX	29	I	**Index:** Signal from the drive indicating the beginning of a track. It is used by the 82062 during formatting, and for counting retries. Index is edge triggered. Only the rising edge is valid.
WR FAULT	30	I	**Write Fault:** An error input to the 82062 which indicates a fault condition at the drive. If WR FAULT from the drive goes high, the command will be terminated.
TRACK 000	31	I	**Track Zero:** Signal from the drive which indicates that the head is at the outermost cylinder. Used by the Restore command.
SC	32	I	**Seek Complete:** Signal from the drive indicating to the 82062 that the drive head has settled and that reads or writes can be made. SC is edge triggered. Only the rising edge is valid.
RWC	33	O	**Reduced Write Current:** Signal goes high for all cylinder numbers above the value programmed in the Write Precomp Cylinder register. It is used by the precompensation logic and by the drive to reduce the effects of bit shifting.
DRUN	34	I	**Data Run:** This signal informs the 82062 when a field of ones or zeroes has been detected in the read data stream by an external one-shot. This indicates the beginning of an ID field. RD GATE is brought high when DRUN is sampled high for 16 clock periods. See note 2.
BRDY	35	I	**Buffer Ready:** Input used to signal the controller that the buffer is ready for reading (full), or writing (empty), by the host uP. Only the rising edge indicates the condition.
BDRQ	36	O	**Buffer Data Request:** Activated during Read or Write commands when a data transfer between the host and the 82062's sector buffer is required. Typically used as a DMA request line, or to generate an interrupt.
RD DATA	37	I	**Read Data:** Single ended input that accepts MFM data from the drive. See note 2.
RD GATE	38	O	**Read Gate:** Output that is high for data and ID fields. Goes active when DRUN has been high for 16 WR CLOCK periods to permit the external phase lock loop to lock onto the incoming disk data stream.
RD CLOCK	39	I	**Read Clock:** Clock input derived from the external data recovery circuits. See note 2.
V_{CC}	40	I	**D.C. Power:** +5V

Note 1: This pin requires a pull-up resistor to function properly. A value of 1000 ohms will work satisfactorily.

Note 2: This pin requires input levels that are not TTL compatible. These lines can be interfaced to TTL with a pull-up resistor. Too small of a resistor will produce a VIL level that is too high. Too large of a resistor will degrade the signal's rise time. A minimum value for the resistor is determined as follows:

$$\frac{(V_{CC}\ \text{max}) - (82062\ V_{IL}\ \text{max})}{(\text{TTL}\ I_{OL}\ \text{min}) - (82062\ I_{IL}\ \text{max})}$$

This would typically be:

$$\frac{5.25\text{V} - 0.5\text{V}}{1.6\ \text{mA} - 10\ \mu\text{A}} \simeq 3\text{k}\ \Omega$$

FUNCTIONAL DESCRIPTION

The Intel 82062 Winchester Disk Controller (WDC) integrates much of the logic needed to implement Winchester Disk controller subsystems. It provides MFM-encoded data and all the control lines required by hard disks using the Seagate Technology ST506 or Shugart Associates SA1000 interface standard. Currently, most 5¼ inch and many 8 inch Winchester Drives use this interface.

Due to the higher data rates required by these drives—1 byte every 1.6 usec—the 82062 is designed to interface with the host CPU or I/O controller through an external buffer RAM. The 82062 WDC has four pins that minimize the logic required to design a buffer interface.

Figure 3 shows a block diagram of an 82062 subsystem. The WDC is controlled by the host CPU through six commands:

Restore
Seek
Read Sector
Write Sector
Scan ID
Write Format

These commands use information stored by six task registers. Command execution starts immediately after the command register is loaded—therefore commands require only one byte from the CPU after the WDC has been initialized.

The 82062 adds all the required track formatting to the data field, including two bytes of CRC. Optionally, these two bytes can be replaced by seven bytes of ECC information for external error correction.

INTERNAL ARCHITECTURE

The internal architecture of the 82062 WDC is shown in more detail in Figure 4. The major functional blocks are:

PLA Controller

The PLA interprets commands and provides all control functions. It is synchronized with WR CLOCK.

Magnitude Comparator

A 10-bit magnitude comparator is used to calculate the direction and number of step pulses needed to move the head from the present to the desired cylinder.

CRC Logic

Generates and checks the cyclic redundancy check characters appended to the ID and data fields. The polynomial used is:

$$X^{16} + X^{12} + X^5 + 1.$$

MFM Encode/Decode

Encodes and decodes MFM data to be written/read from the drive. The MFM encoder operates from WR CLOCK, a clock having a frequency equivalent to the bit rate. The MFM decoder operates from RD CLOCK, a bit rate clock generated from the external data separator. RD CLOCK and WR CLOCK need not be synchronized.

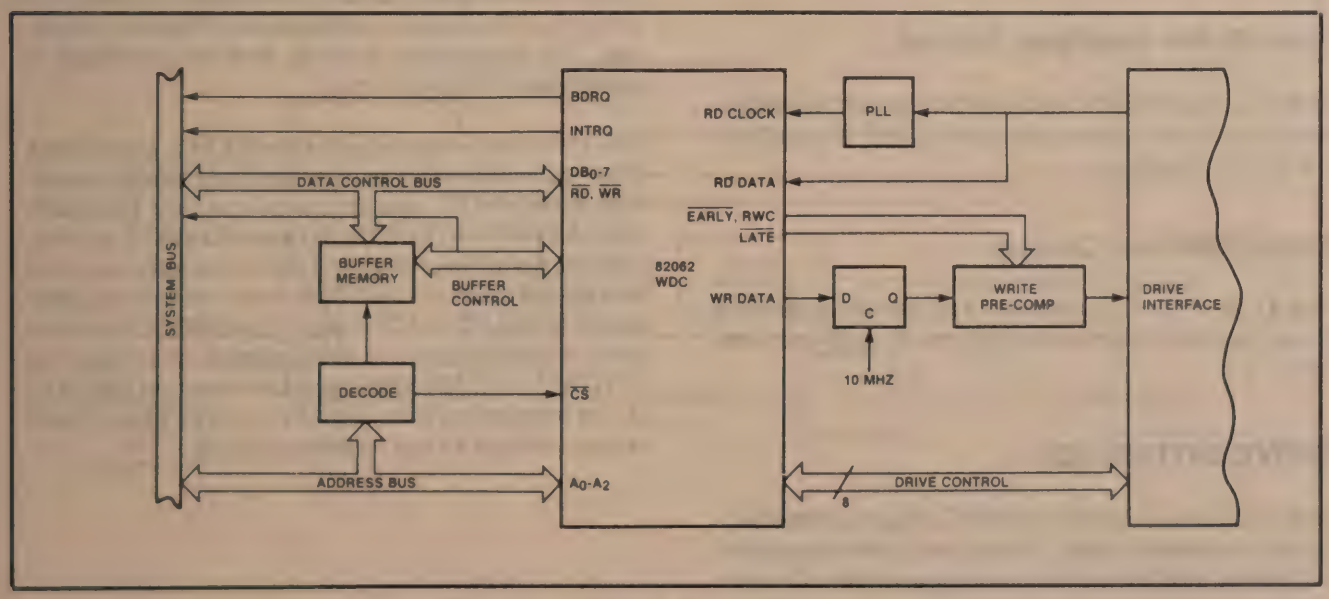


Figure 3. System Block Diagram

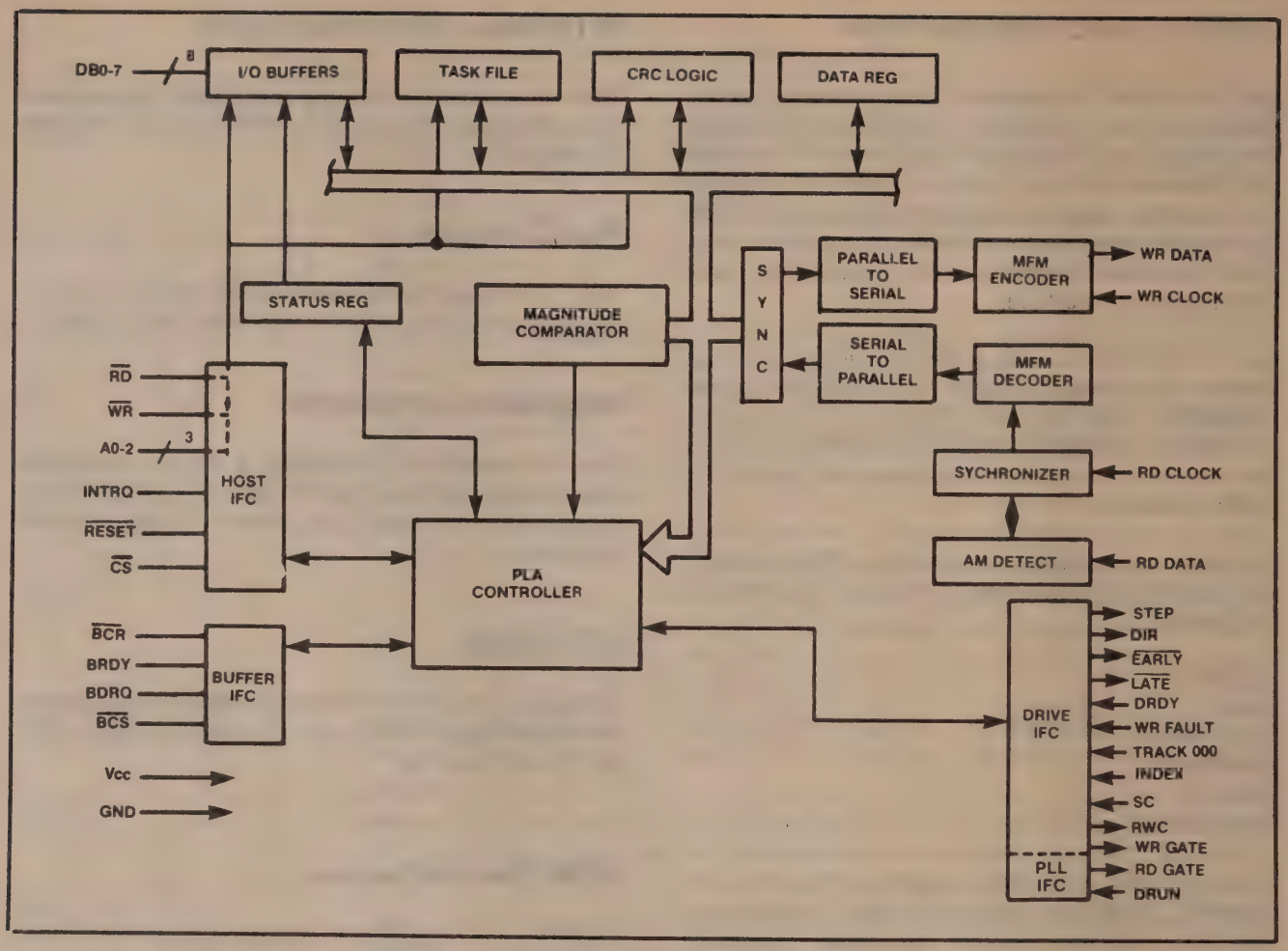


Figure 4. 82062 Detailed Block Diagram

AM Detect

The address mark detector checks the incoming data stream for a unique missing clock pattern (Data = A1H, Clock = 0AH) used in each ID and data field.

Host/Buffer Interface Control

The Host/Buffer IFC logic contains all of the necessary circuitry to communicate with the 8-bit bus from the host processor.

Drive Interface Control

The Drive IFC logic controls and monitors all lines from the drive, with the exception of read and write data.

DRIVE INTERFACE

The drive side of the 82062 WDC requires three sections of external logic. These are buffer/receivers, data separator, and write precompensation. Figure 5 illustrates a drive side interface.

The buffer/receivers condition the control lines to be driven down the cable to the drive. The control lines are typically single-ended, resistor terminated TTL levels. The data lines to and from the drive also require buffering, but are differential RS-422 levels. The interface specification to the drive can be found in the manufacturers' OEM manual. The WDC supplies TTL compatible signals, and will interface to most buffer/driver devices.

The data recovery circuits consist of a phase-lock loop data separator and associated components. The 82062 WDC interacts with the data separator thru the DATA RUN (DRUN) and RD GATE signals. A block diagram of a typical data separator circuit is shown in Figure 6. Read data from the drive is presented to the RD DATA input of the WDC, the reference multiplexor, and a retriggerable one-shot. The RD GATE (Pin 38) output will be low when the WDC is not inspecting data. The PLL at this time should remain locked to the reference clock.

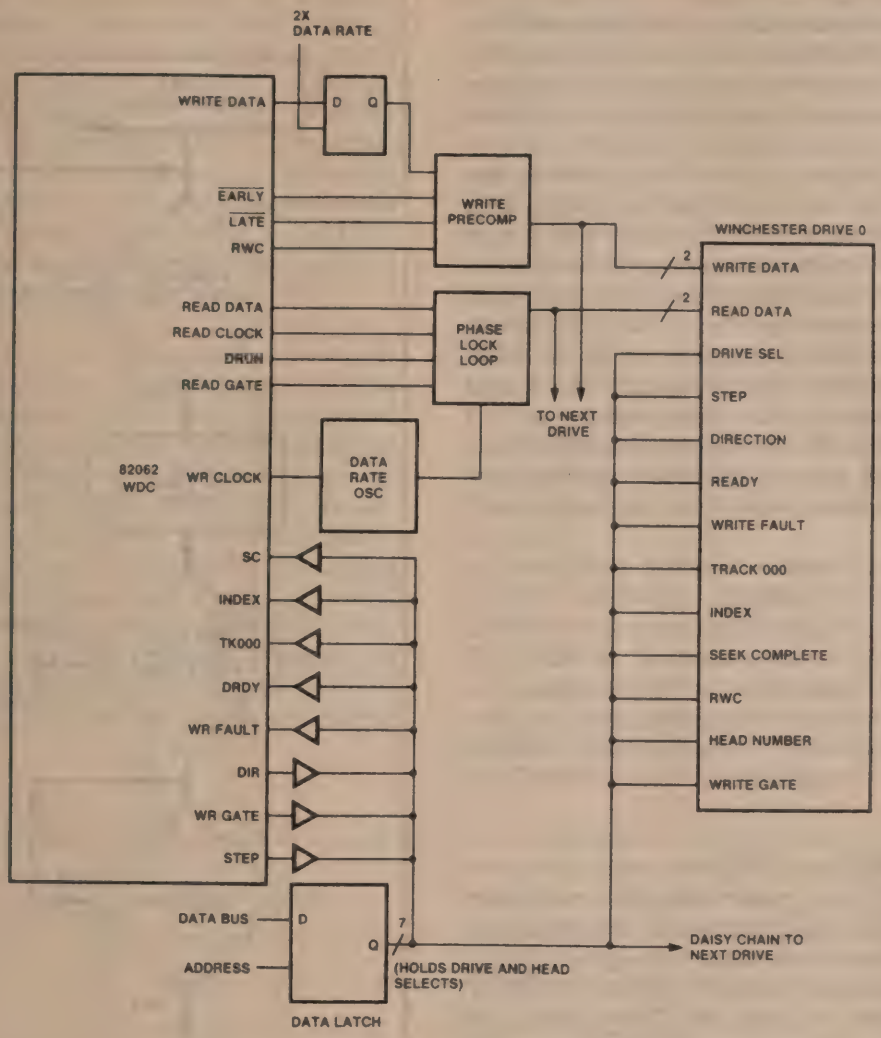


Figure 5. Drive Interface

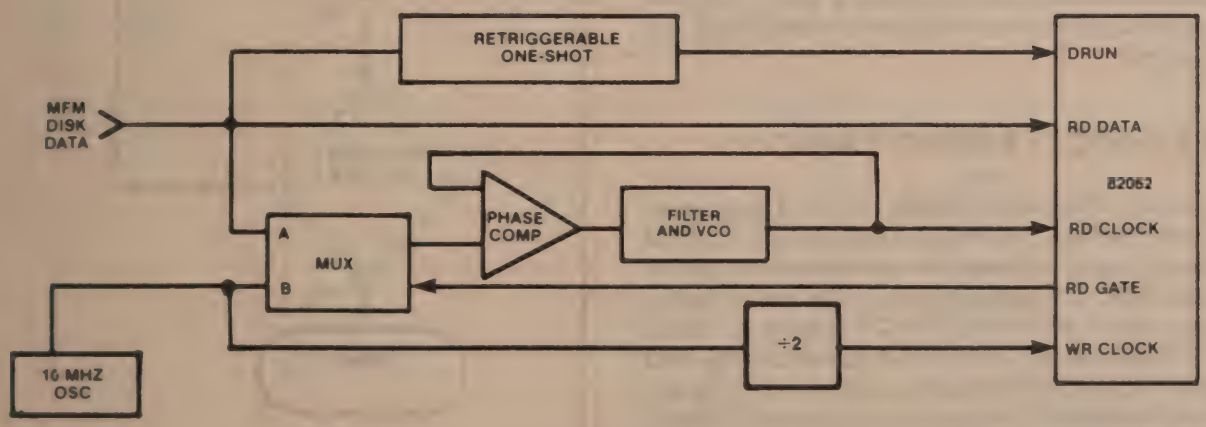


Figure 6. Data Recovery Circuit

When any Read/Write command is initiated and a search for address mark begins, the DRUN input is examined. The DRUN one-shot is set for slightly greater than one bit time, allowing it to retrigger constantly on a field of ones and zeros. An internal counter times out to see that DRUN is high for 2 byte times. RD GATE is set by the WDC, switching the data separator to lock onto the incoming data stream. If DRUN falls prior to an additional 7 byte times, RD GATE is lowered and the process is repeated. RD GATE will remain active high until a non-zero, non-address mark byte is detected. It will then lower RD GATE for two byte times (to allow the PLL to lock back on to the reference clock), and start the DRUN search again. If an address mark is detected, RD GATE will be held high and the command will continue searching for the proper ID field. This sequence is shown in the flow chart in Figure 7.

The write precompensation logic is controlled by the signals REDUCE WRITE CURRENT (RWC), $\overline{\text{EARLY}}$ and $\overline{\text{LATE}}$. The cylinder in which the RWC line becomes active is controlled by the REDUCE WRITE CURRENT register in the Task Register File. It can be used to turn on the precomp circuitry on a predetermined cylinder. If the REDUCE WRITE CURRENT register contents are FFH, then RWC will always be low.

The signals $\overline{\text{EARLY}}$ and $\overline{\text{LATE}}$ are used to tell the precomp circuitry how much delay is required on the WR DATA pulse about to be sent. The amount of delay is determined externally through a digital delay line or equivalent circuitry. Since the $\overline{\text{EARLY}}$ signal occurs after the fact, WR DATA should be delayed by one interval when both $\overline{\text{EARLY}}$ and $\overline{\text{LATE}}$ are deasserted, two intervals when $\overline{\text{LATE}}$ is asserted, and no delay when $\overline{\text{EARLY}}$ is asserted. An interval is, for example, 12-15 ns. for the ST506 interface. $\overline{\text{EARLY}}$ or $\overline{\text{LATE}}$ will be active slightly ahead of the WR DATA pulse. $\overline{\text{EARLY}}$ and $\overline{\text{LATE}}$ will never be asserted at the same time. $\overline{\text{EARLY}}$ and $\overline{\text{LATE}}$ are always active, and should be gated externally by the RWC signal.

HOST PROCESSOR INTERFACE

The primary interface between the host processor and the 82062 WDC is through an 8-bit bi-directional data bus. This bus is used to transmit/receive data to both the WDC and a sector buffer. The sector buffer is constructed with either FIFO memory, or static RAM and a counter. Since the WDC will use the data bus when accessing the sector buffer, a transceiver must be used to isolate the host during this time. Figure 8 shows a typical connection to a sector buffer implemented with RAM memory. Whenever the WDC is not using the sector buffer, The BUFFER CHIP SELECT ($\overline{\text{BCS}}$) is high (disabled). This allows the host to access the WDC's Task Register File, and

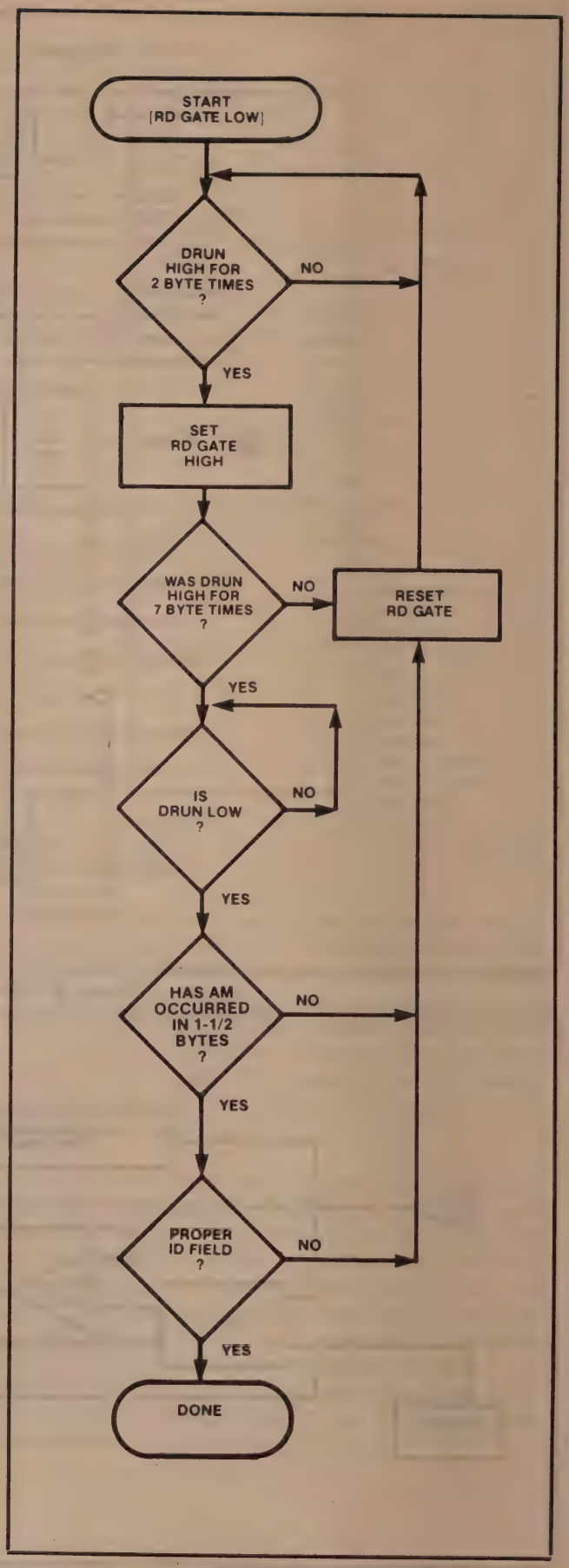


Figure 7. PLL Control Sequence

to set up parameters prior to issuing a command. It also allows the host to access the RAM buffer. A decoder is used to generate a chip select when A_{0-2} is '000', an unused address in the WDC. A binary counter is enabled whenever $\overline{RD}$ or $\overline{WR}$ go active and is incremented on the trailing edge of the chip select. This allows the host to access sequential bytes within the RAM. The decoder also generates another chip select when A_{0-2} does not equal '000', allowing access to the WDC's internal registers while keeping the RAM tri-stated.

During a WRITE SECTOR command, the host processor sets up data in the Task Register File and then issues the command. It then generates a status to inform the host that it may load the buffer with the data to be written. When the counter reaches its maximum count, the BUFFER READY (BRDY) signal is made active (by the "carry" out of the counter), informing the WDC that the buffer is full. (BRDY is a rising edge triggered signal which will be ignored if activated before the WDC issues $\overline{BCR}$). $\overline{BCS}$ is then made active, disconnecting the host through the transceivers, and the $\overline{RD}$ and $\overline{WR}$ lines become outputs from the WDC to allow it to access the buffer.

When the WDC is done using the buffer, it disables $\overline{BCS}$ which again allows the host to access the local bus. The READ SECTOR command operates in a similar manner, except the buffer is loaded by the WDC instead of the host processor.

Another control signal called BUFFER DATA REQUEST (BDRQ, not used in Figure 8) is a DMA signal that can inform a DMA controller when the 82062 WDC is requesting data. For further explanation, refer to the individual command descriptions and the A.C. Characteristics. In a READ SECTOR command, interrupts are generated at the termination of the command. An interrupt may be specified to occur either at the end of the command, or when BDRQ is activated. The INTERRUPT line (INTRQ) is cleared either by reading the STATUS register, or by writing a new command in the COMMAND register.

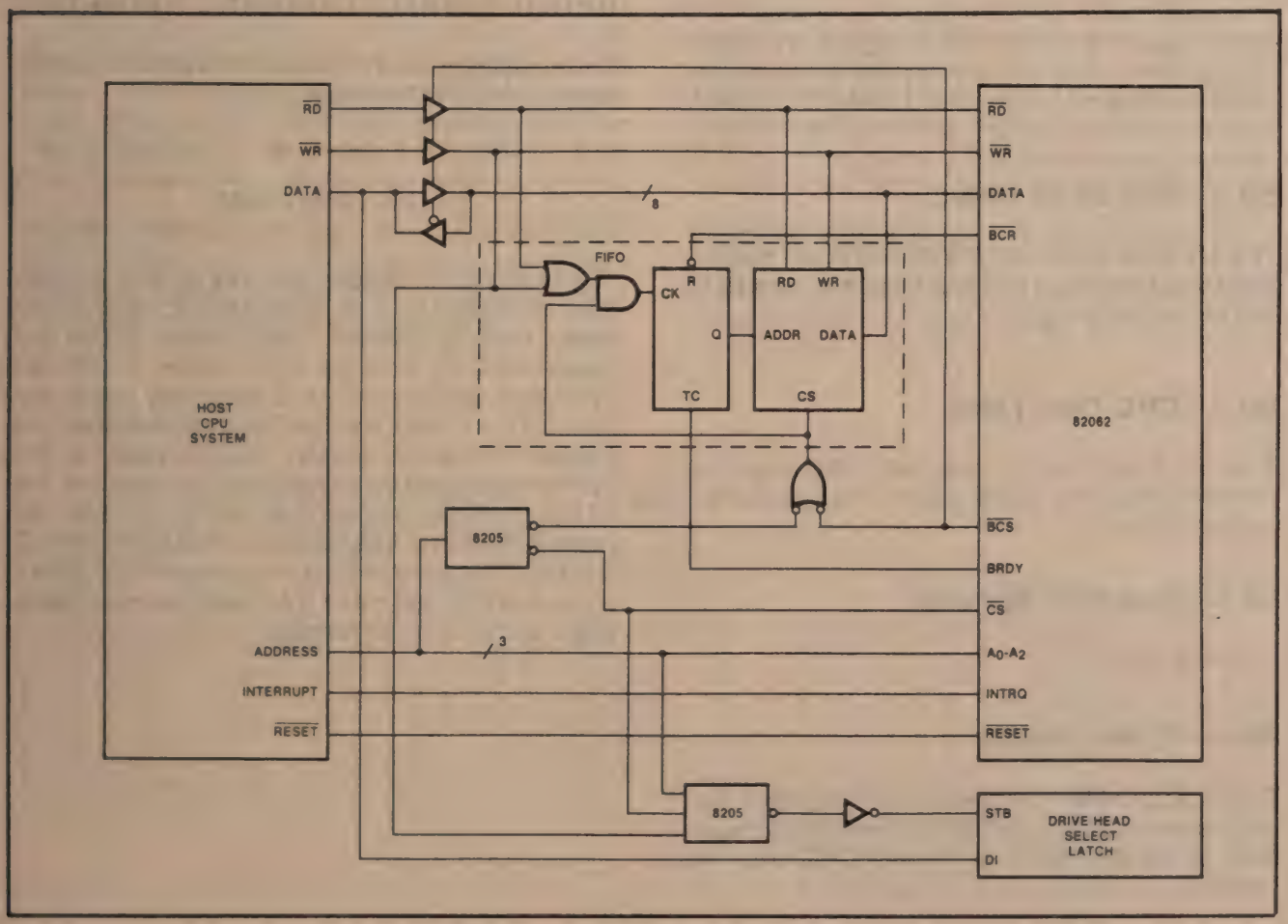


Figure 8. CPU Buffer Interface

TASK REGISTER FILE

The Task Register File is a bank of registers used to hold parameter information pertaining to each command. These registers and their addresses are:

A2	A1	A0	READ	WRITE
0	0	0	(Bus Tri-Stated)	(Bus Tri-Stated)
0	0	1	Error Flags	Reduce Write Current
0	1	0	Sector Count	Sector Count
0	1	1	Sector Number	Sector Number
1	0	0	Cylinder Low	Cylinder Low
1	0	1	Cylinder High	Cylinder High
1	1	0	SDH	SDH
1	1	1	Status Register	Command Register

NOTE: Registers are not cleared by $\overline{\text{RESET}}$.

ERROR REGISTER

This read-only register contains specific error status after the completion of a command. The bits are defined as follows:

7	6	5	4	3	2	1	0
BBD	CRC	—	ID	—	AC	TK000	DM

Bit 7 - Bad Block Detect

This bit is set when an ID field has been encountered that contains a bad block mark. It is used for bad sector mapping.

Bit 6 - CRC Data Field

This bit is set when a data field CRC error has ocurred. The sector buffer may still be read but will contain errors.

Bit 5 - Reserved Not used.

Forced to zero.

Bit 4 - ID Not Found

This bit is set when the desired cylinder, head, sector, or size parameter cannot be found after 8 revolutions of the disk, or if an ID field CRC error has occured.

Bit 3 - Reserved Not used.

Forced to zero.

Bit 2 - Aborted Command

This bit is set if a command was issued while DRDY (Pin 28) is deasserted or WR FAULT (Pin 30) is asserted. The Aborted Command bit will also be set if an undefined command is written into the COMMAND register, but an implied seek will be executed.

Bit 1 - TRACK 000

This bit is set only by the RESTORE command. It indicates that TRACK 000 (Pin 31) has not gone active after the issuance of 1024 stepping pulses.

Bit 0 - Data Address Mark

This bit is set during a READ SECTOR command if the Data Address Mark is not found after the proper Sector ID is read.

REDUCE WRITE CURRENT REGISTER

This register is used to define the cylinder number where RWC (Pin 33) is asserted:

7	6	5	4	3	2	1	0
CYLINDER NUMBER ÷ 4							

The value (0-255) loaded into this register is internally multiplied by 4 to specify the actual cylinder where RWC is asserted. Thus a value of 01H will cause RWC to activate on cylinder 4, 02H on cylinder 8, and so on. RWC switching points are then 0,4,8, . . . 1020. RWC will be asserted when the present cylinder is greater than or equal to the cylinder indicated by this register. For example, the ST506 interface requires precomp on cylinder 128 (80H) and above. Therefore, the REDUCE WRITE CURRENT register should be loaded with 32 (20H). A value of FFH will make RWC stay low, regardless of the actual cylinder number.

SECTOR COUNT REGISTER

This register is used to define the number of sectors that need to be transfered to the buffer during a READ MULTIPLE SECTOR or WRITE MULTIPLE SECTOR command.:

7	6	5	4	3	2	1	0
# OF SECTORS							

The value contained in the register is decremented after each sector is transferred to/from the sector buffer. A zero represents a 256 sector transfer, a one a 1 sector transfer, etc. This register is a "don't care" when single sector commands are specified.

SECTOR NUMBER

This register holds the sector number of the desired sector:

7	6	5	4	3	2	1	0
SECTOR NUMBER							

For a multiple sector command, it specifies the first sector to transferred. It is incremented after each sector is transferred to/from the sector buffer. The SECTOR NUMBER register may contain any value from 0 to 255.

The SECTOR NUMBER register is also used to program the Gap 1 and Gap 3 lengths to be used when formatting a disk. See the WRITE FORMAT command description for further explanation.

CYLINDER NUMBER LOW REGISTER

This register holds the lower byte of the desired cylinder number:

7	6	5	4	3	2	1	0
LS BYTE OF CYLINDER NUMBER							

It is used in conjunction with the CYLINDER NUMBER HIGH register to specify a range of 0 to 1023.

CYLINDER NUMBER HIGH REGISTER

This register holds the two most significant bits of the desired cylinder number:

7	6	5	4	3	2	1	0
X	X	X	X	X	X	(9)	(8)

Internal to the 82062 WDC is another pair of registers that hold the actual position where the R/W heads are located. The CYLINDER NUMBER HIGH and LOW registers can be considered the cylinder destination for seeks and other commands. After these commands are executed, the internal cylinder position registers' contents are equal to the cylinder high/low registers. If a drive number change is detected on a new command, the WDC automatically reads an ID field to update its internal cylinder position registers. This affects all commands except a RESTORE.

SECTOR/DRIVE/HEAD REGISTER

The SDH register contains the desired sector size, drive number, and head number parameters. The format is diagramed below.

7	6	5	4	3	2	1	0
EXT	SIZE		DRIVE		HEAD		

6	5	SECTOR SIZE
0	0	256
0	1	512
1	0	1024
1	1	128

4	3	DRIVE #
0	0	DSEL1
0	1	DSEL2
1	0	DSEL3
1	1	DSEL4

2	1	0	HEAD #
0	0	0	HSEL0
0	0	1	HSEL1
0	1	0	HSEL2
0	1	1	HSEL3
1	0	0	HSEL4
1	0	1	HSEL5
1	1	0	HSEL6
1	1	1	HSEL7

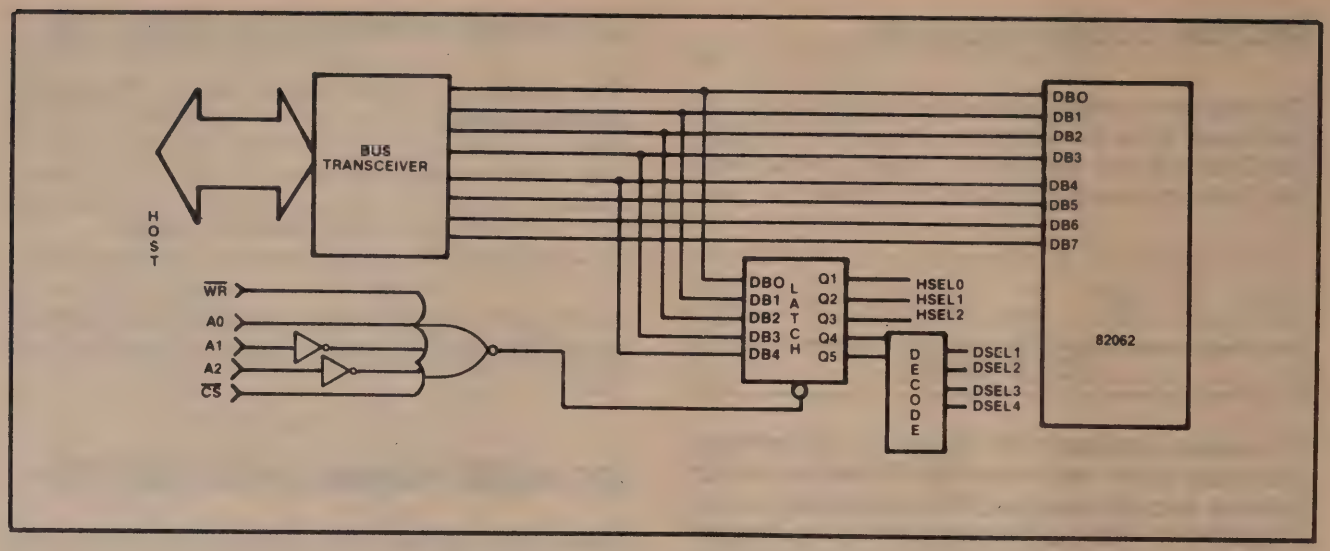


Figure 9. Drive/Head Select Logic

Both head number and sector size are compared against the disks' ID field. Head select and drive select lines are not available as outputs from the 82062 WDC and must be generated externally. Figure 9 shows a possible logic implementation of these select lines.

Bit 7, the extension bit (EXT), is used to extend the data field by seven bytes when using ECC codes. When EXT = 1, the CRC is not appended to the end of the data field, the data field becomes "sector size + 7" bytes long. The CRC is checked on the ID field regardless of the state of EXT. Note that the sector size bits (SIZE) are written to the ID field during a formatting command. The SDH byte written into the ID field is different than the SDH Register contents. The recorded SDH byte does not have the drive number (DRIVE) written but does have the BAD BLOCK mark written. The format is:

7	6	5	4	3	2	1	0
BAD BLOCK	SIZE		0	0		HEAD #	

Note that use of the extension bit requires the gap lengths to be modified as described in the WRITE FORMAT command description.

STATUS REGISTER

The status register is a read-only register which informs the host of certain events performed by the 82062 WDC as well as reporting status from the drive control lines. The INTRQ line will be reset when the status register is read. The format is:

7	6	5	4	3	2	1	0
BUSY	READY	WF	SC	DRQ	—	CIP	ERROR

Bit 7 - Busy

This bit is set whenever the 82062 WDC is accessing the disk. Commands should not be loaded into the COMMAND register while Busy is set. Busy is set when a command is written into the WDC and is cleared at the end of all commands except READ SECTOR. While executing a READ SECTOR command, Busy is cleared after the sector buffer has been filled. When the Busy bit is set, no other bits in either the STATUS or any other registers are valid.

Bit 6 - Ready

This bit reflects the state of the DRDY (Pin 28) line.

Bit 5 - Write Fault

This bit reflects the state of the WR FAULT (Pin 30) line. Whenever WR FAULT goes high, an interrupt will be generated.

Bit 4 - Seek Complete

This bit reflects the state of the SC (Pin 32) line. Commands which initiate a seek will pause until Seek Complete is set.

Bit 3 - Data Request

The Data request bit (DRQ) reflects the state of the BDRQ (Pin 36) line. It is set when the sector buffer should be loaded with data or read by the host processor, depending upon the command. The DRQ bit and the BDRQ line remain high until BRDY is sensed, indicating the operation is completed. BDRQ can be used in DMA interfacing, while DRQ can be used for programmed I/O transfers.

Bit 2 - Reserved

Not Used. Forced to zero.

Bit 1 - Command in Progress

When this bit is set, a command is being executed and a new command should not be loaded until it is cleared. Although a command may be executing, the sector buffer is still available for access by the host processor. Only the STATUS register may be read. If other registers are read, the STATUS register contents will be returned.

Bit 0 - Error

This bit is set whenever any bits in the ERROR register are set. It is the logical 'or' of the bits in the error register and may be used by the host processor to quickly check for successful completion of a command. This bit is reset when a new command is written into the COMMAND register.

COMMAND REGISTER

This write-only register is loaded with the desired command:

7	6	5	4	3	2	1	0
COMMAND							

The command begins to execute immediately upon loading. This register should not be loaded while the Busy or Command in Progress bits are set in the STATUS register. The INTRQ line (Pin 3), if set, will be cleared by a write to the COMMAND register.

INSTRUCTION SET

The 82062 WDC instruction set contains six commands. Prior to loading the command register, the host processor must first set up the Task Register File with the information needed for the command. Except for the COMMAND register, the registers may be loaded in any order. If a command is in progress, a subsequent write to the COMMAND register will be ignored until execution of the current command is completed as indicated by the command in progress bit in the STATUS register being cleared

COMMAND	7	6	5	4	3	2	1	0
RESTORE	0	0	0	1	R3	R2	R1	R0
SEEK	0	1	1	1	R3	R2	R1	R0
READ SECTOR	0	0	1	0	I	M	0	T
WRITE SECTOR	0	0	1	1	0	M	0	T
SCAN ID	0	1	0	0	0	0	0	T
WRITE FORMAT	0	1	0	1	0	0	0	0

R_{3-0} = Rate Field

For 5 MHz WR CLOCK:

R_{3-0} =	0000	- ≈35	us
	0001	- 0.5	ms
	0010	- 1.0	ms
	0011	- 1.5	ms
	0100	- 2.0	ms
	0101	- 2.5	ms
	0110	- 3.0	ms
	0111	- 3.5	ms
	1000	- 4.0	ms
	1001	- 4.5	ms
	1010	- 5.0	ms
	1011	- 5.5	ms
	1100	- 6.0	ms
	1101	- 6.5	ms
	1110	- 7.0	ms
	1111	- 7.5	ms

T = Retry Enable

T = 0 **Enable Retries**
T = 1 **Disable Retries**

M = Multiple Sector Flag

M = 0 **Transfer 1 Sector**
M = 1 **Transfer Multiple Sectors**

I = Interrupt Enable

I = 0 **Interrupt at BDRQ time**
I = 1 **Interrupt at end of command**

RESTORE COMMAND

The RESTORE command is usually used on a power-up comdition. The actual stepping rate used for the RESTORE is determined by the Seek Complete time. A step pulse is issued and the 82062 WDC waits for a rising edge on the Seek Complete (SC) line before issuing the next pulse. If 10 index pulses are received without a rising edge of SC, the 82062 will switch to sensing the level of the SC line. If after 1,024 stepping pulses the TRACK 000 line does not go active, the WDC will set the TRACK 000 bit in the ERROR register and terminate with an INTRQ. An interrupt will also occur if WR FAULT goes active or DRDY goes inactive at any time during execution.

The rate field specified R_{3-0} is stored in an internal register for future use in commands with implied seeks.

A flowchart of the RESTORE command is shown in Figure 10.

SEEK COMMAND

Since all commands except the SCAN ID command feature an implied seek, the SEEK command can be used for overlap seek operations on multiple drives. The actual stepping rate used is taken from the Rate Field of the command, and is stored in an internal register for future use. If DRDY goes inactive or WR FAULT goes active at any time during the seek, the command is terminated and an INTRQ is generated.

The direction and number of step pulses needed is calculated by comparing the contents of the CYLINDER NUMBER LOW/HIGH register pair to the internal cylinder position register. After all steps have been issued, the internal cylinder position register is updated and the command is terminated. The Seek Complete (SC) line is not checked at the beginning or end of the command.

If an implied seek was performed, the 82062 will search until a rising edge of SC is received. If 10 index pulses are received without a rising edge of SC, the 82062 will switch to sensing the level of the SC line.

A flowchart of the SEEK command is shown in Figure 11.

READ SECTOR

The READ SECTOR command is used to transfer one or more sectors of data from the disk to the sector buffer. Upon receipt of the READ SECTOR command, the 82062 WDC checks the CYLINDER NUMBER LOW/HIGH register pair against the internal cylinder position register to see if they are equal. If not, the direction and number of steps calculation is performed and a seek takes place. If an implied seek was performed, the WDC will search until a rising edge of SC is received. The WR FAULT and DRDY lines are monitored throughout the command.

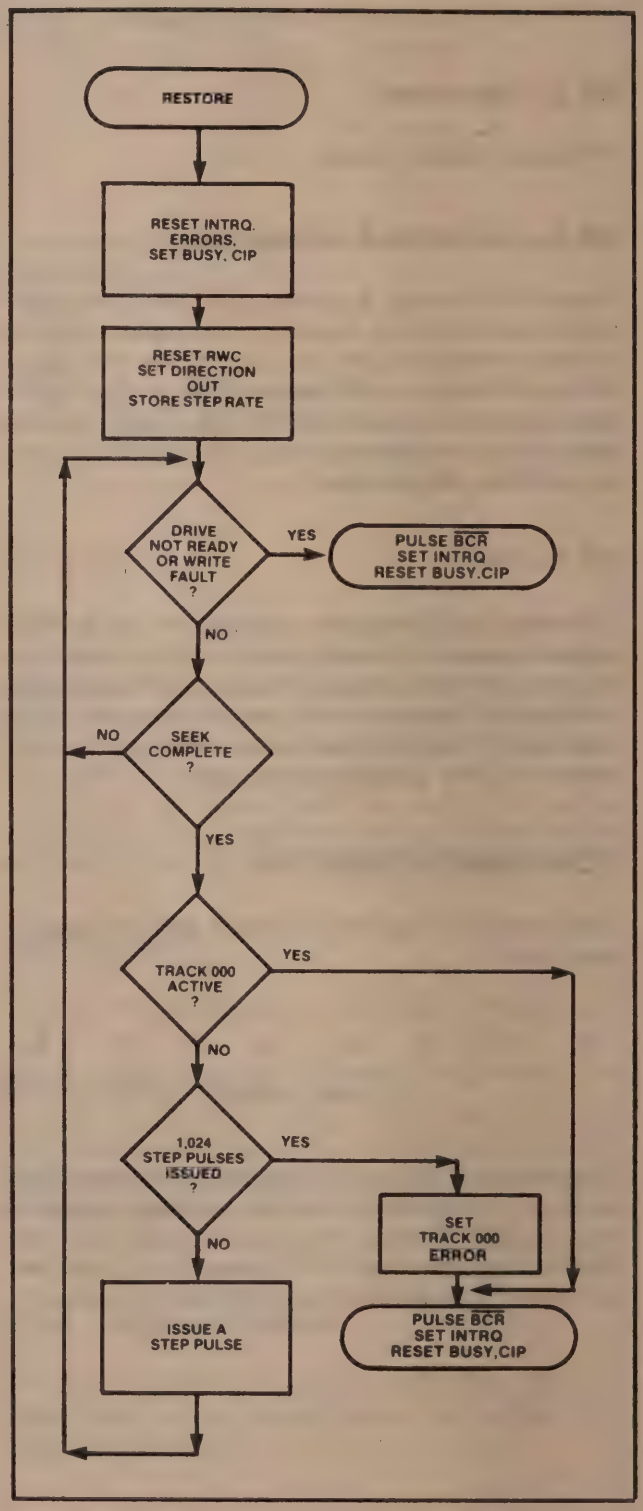


Figure 10. Restore Command Flow

When the Seek Complete (SC) line is high (with or without an implied seek having occured), the search for an ID field begins. If T = 0 (retries enabled), the 82062 WDC must find an ID with the correct cylinder number, head, sector size and CRC within 10 revolutions, or an automatic scan ID will be performed to obtain cylinder position information, and then a seek performed (if necessary). The search for the proper ID will be retried for up to 10 revolutions. If the correct sector is still not found, the appropriate error bits will be set and the command terminated. Data CRC errors will also be retried for up to 10 revolutions (if T = 0).

If T = 1 (retries disabled), the ID search must find the correct sector within 2 revolutions or the appropriate error bits will be set and the command terminated.

Both the READ SECTOR and WRITE SECTOR commands feature a "simulated completion" to ease programming. DRQ/BDRQ will be generated upon detecting an error condition. This allows the same program flow for successful or unsuccessful completion of a command.

When the data address mark is found, the WDC is ready to transfer data to the sector buffer. After the data has been transferred, the I bit is checked. If I = 0, INTRQ is made active coincident with BDRQ, indicating that a transfer of data from the buffer to the host processor is required. If I = 1, INTRQ will occur at the end of the command, i.e. after the buffer is unloaded by the host.

An optional M bit may be set for multiple sector transfers. When M = 0, one sector is transferred and the SECTOR COUNT register is ignored. When M = 1, multiple sectors are transferred. After each sector is transferred the 82062 decrements the SECTOR COUNT register and increments the SECTOR NUMBER register. The next logical sector will be transferred regardless of any interleave. Sectors are numbered at format time by a byte in the ID field.

For the 82062 to make multiple sector transfers to the buffer, the BRDY line must be toggled low to high for each sector. Transfers will continue until the SECTOR COUNT register equals zero, or the BRDY line goes active. If the SECTOR COUNT register is nonzero (indicating more sectors are to be transferred but the buffer is full), BDRQ will be made active and the host must unload the buffer. After this occurs, the buffer will again be free to accept the remaining sectors from the WDC. This scheme enables the user to transfer more sectors than the buffer memory has capacity for.

In summary then, READ SECTOR operation is as follows:

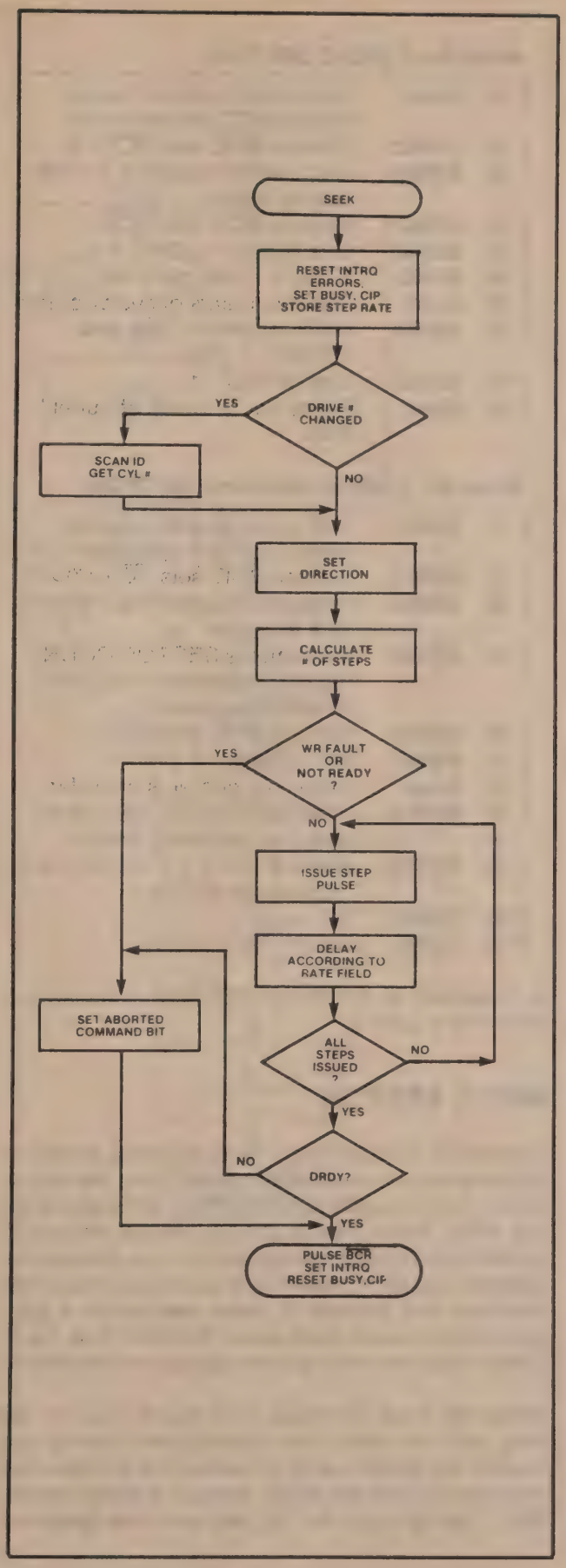


Figure 11. Seek Command Flow

When M = 0 (READ SECTOR)

(1)	**Host:**	Sets up parameters; issues READ SECTOR command.
(2)	**82062:**	Strobes $\overline{BCR}$; sets $\overline{BCS}$ = 0.
(3)	**82062:**	Finds sector specified; transfers data to buffer.
(4)	**82062:**	Strobes $\overline{BCR}$; sets $\overline{BCS}$ = 1.
(5)	**82062:**	Sets BDRQ = 1, DRQ = 1.
(6)	**82062**	If I bit = 1 then go to (9).
(7)	**Host:**	Reads contents of sector buffer.
(8)	**82062:**	Waits for BRDY, then sets INTRQ = 1; END.
(9)	**82062:**	Sets INTRQ = 1.
(10)	**Host:**	Reads out contents of buffer; END.

When M = 1 (READ MULTIPLE SECTOR)

(1)	**Host:**	Sets up parameters; issues READ SECTOR command.
(2)	**82062:**	Strobes $\overline{BCR}$; sets $\overline{BCS}$ = 0.
(3)	**82062:**	Finds sector specified; transfers data to buffer.
(4)	**82062:**	Decrements SECTOR COUNT register; increments SECTOR NUMBER register.
(5)	**82062:**	Strobes $\overline{BCR}$; sets $\overline{BCS}$ = 1.
(6)	**82062:**	Sets BDRQ = 1, DRQ = 1.
(7)	**Host:**	Reads out contents of buffer;
(8)	**Buffer:**	Indicates data has been transferred by activating BRDY.
(9)	**82062:**	When BRDY = 1, if Sector Count = 0, then go to (11).
(10)	**82062:**	Go to (2).
(11)	**82062:**	Set INTRQ = 1; END.

A flowchart of the READ SECTOR command is shown in Figure 12.

WRITE SECTOR

The WRITE SECTOR command is used to write one or more sectors of data to the disk from the sector buffer. Upon receipt of WRITE SECTOR command, the 82062 WDC checks the CYLINDER NUMBER LOW/HIGH register pair against the internal cylinder position register to see if they are equal. If not, the direction and number of steps calculation is performed and a seek takes place. The WR FAULT and DRDY lines are checked throughout the command.

When the Seek Complete (SC) line is found to be true (with or without an implied seek having occured), the BDRQ signal is made active and the host proceeds to load the buffer. When the 82062 senses BRDY going high, the ID field with the specified cylinder number, head, and sector size is searched for. Once found, WR GATE is made active and the data is written to the disk. It is necessary to resynchronize the write data since a bit cell can extend from 295 nS to 315 nS during a write cycle. If retries are enabled (T = 0), and if the ID field cannot be found within 10 revolutions, automatic scan ID and seek commands are performed. The ID Not Found error bit is set and the command is terminated if the correct ID field is not found within 10 additional revolutions. If retries are disabled, (T = 1), and if the ID field cannot be found within 2 revolutions, the ID Not Found error bit is set and the command is terminated.

During a WRITE MULTIPLE SECTOR command (M = 1), the SECTOR NUMBER register is incremented and the SECTOR COUNT register is decremented. If the BRDY line is asserted after the first sector is transferred from the buffer, the 82062 will transfer the next sector. If BRDY is deasserted, the 82062 will set BDRQ and wait for the host processor to place more data in the buffer. In summary then, the WRITE SECTOR operation is as follows:

When M = 0,1 (WRITE SECTOR)

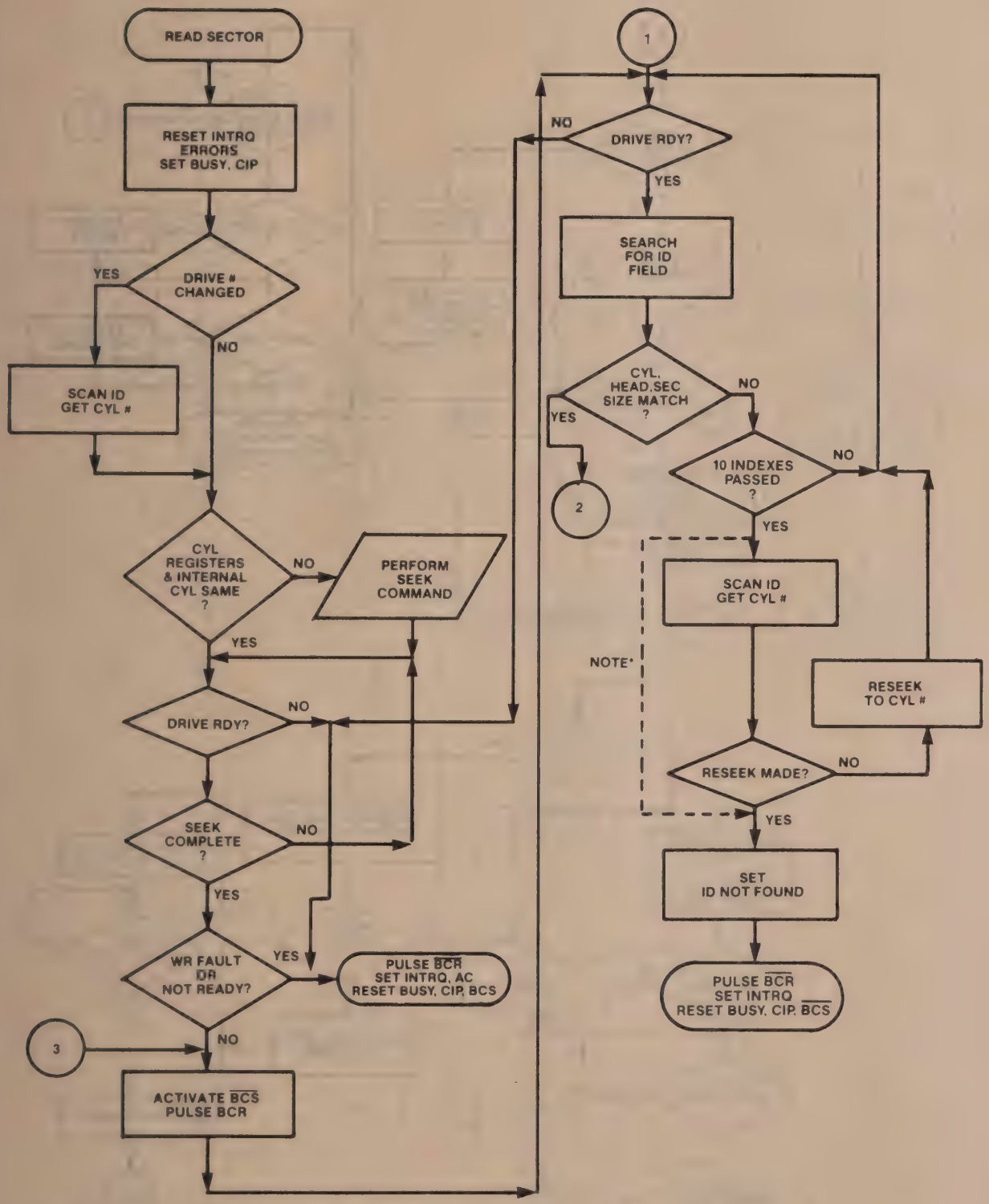
(1)	**Host:**	Sets up parameters; issues WRITE SECTOR command.
(2)	**82062:**	Sets BDRQ = 1, DRQ = 1.
(3)	**Host:**	Loads sector buffer with data.
(4)	**82062:**	Waits for BRDY = low to high.
(5)	**82062:**	Finds specified ID field; writes sector to disk.
(6)	**82062:**	If M = 0, then set INTRQ = 1; END.
(7)	**82062**	Increment SECTOR NUMBER register; decrement SECTOR COUNT register.
(8)	**82062**	If SECTOR = 0, then set INTRQ = 1; END.
(9)	**82062**	Go to (2).

A flowchart of the WRITE SECTOR command is shown in Figure 13.

SCAN ID

The SCAN ID command is used to update the SECTOR/DRIVE/HEAD, SECTOR NUMBER, and CYLINDER NUMBER LOW/HIGH registers.

After the command is loaded, the Seek Complete (SC) line is sampled until it is valid. The DRDY and WR FAULT lines are also monitored throughout execution of the command When the first ID field is



*If T bit of command = 1 then dashed path is taken after 2 index pulses.

Figure 12A. Read Sector Command Flow

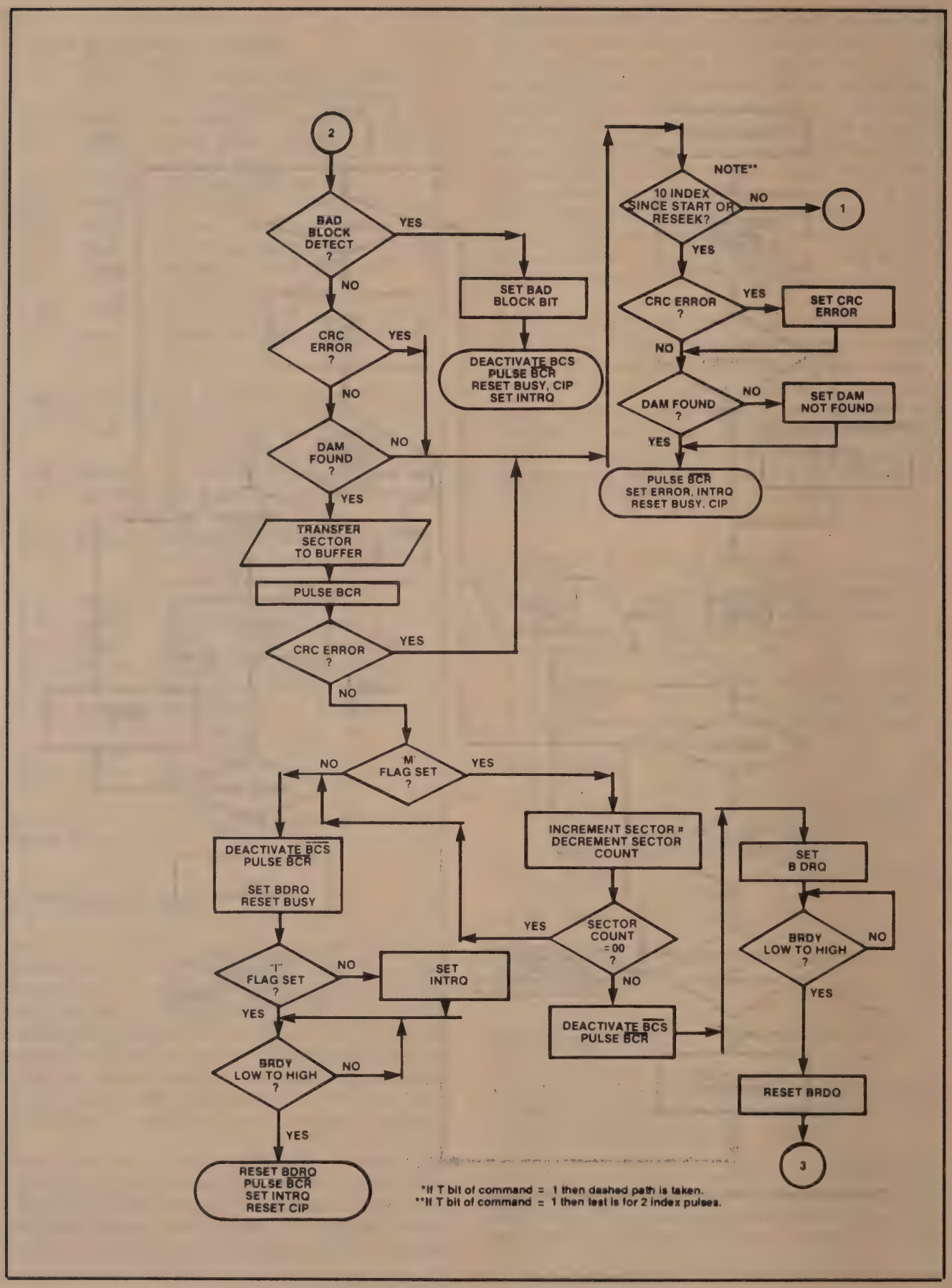


Figure 12B. Read Sector Command Flow

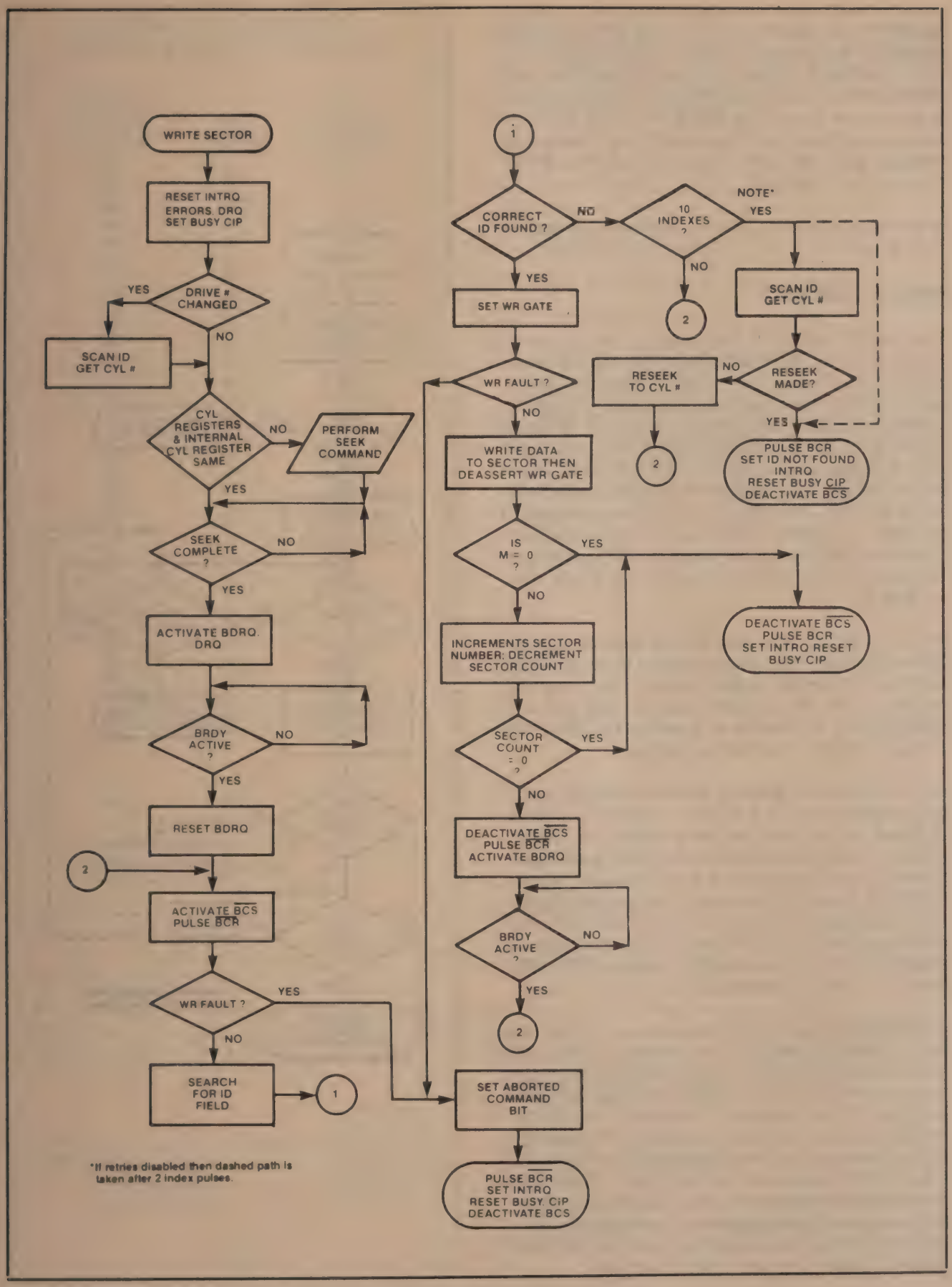


Figure 13. Write Sector Command Flow

found, the ID information is loaded into the SDH, SECTOR NUMBER, and CYLINDER NUMBER registers. The internal cylinder position register is also updated. If a bad block is detected, the BAD BLOCK bit will also be set. The CRC is checked and if an error is found, the 82062 will retry up to 10 revolutions to find an error-free ID field. There is no implied seek with this command and the sector buffer is not disturbed.

A flowchart of the SCAN ID command is shown in Figure 14.

WRITE FORMAT

The WRITE FORMAT command is used to format one track using the Task Register File and the sector buffer. During execution of this command, the sector buffer is used for additional parameter information instead of sector data. Shown in Figure 15 is the contents of the sector buffer for a 32 sector track format with an interleave factor of two. Each sector requires a two byte sequence. The first byte designates whether a bad block mark is to be recorded in the sector's ID field. An 00H is normal; an 80H indicates a bad block mark for that sector. In the example of Figure 15, sector 04 will get a bad block mark recorded.

The second byte indicates the logical sector number to be recorded. This allows sectors to be recorded with any interleave factor desired. The remaining memory in the sector buffer may be filled with any value; its only purpose is to generate a BRDY to tell the 82062 to begin formatting the track.

An implied seek is in effect on this command. As for other commands, if the drive number has been changed, an ID field will be scanned for cylinder position information before the implied seek is performed. If no ID field can be read (because the track had been erased or because an incomplete format had been been used), an ID Not Found error will result and the WRITE FORMAT command will be aborted. This can be avoided by issuing a RESTORE command before formatting.

The SECTOR COUNT register is used to hold the total number of sectors to be formatted (FFH = 255 sectors), while the SECTOR NUMBER register holds the number of bytes minus three to be used for Gap 1 and Gap 3; for instance, if the SECTOR COUNT register value is 02H and the SECTOR NUMBER register value is 00H, then 2 sectors are written and 3 bytes of 4EH are written for Gap 1 and Gap 3. The data fields are filled with FFH and the CRC is automatically generated and appended. The sector extension bit in the SDH register should not be set. After the last sector is written the track is filled with 4EH.

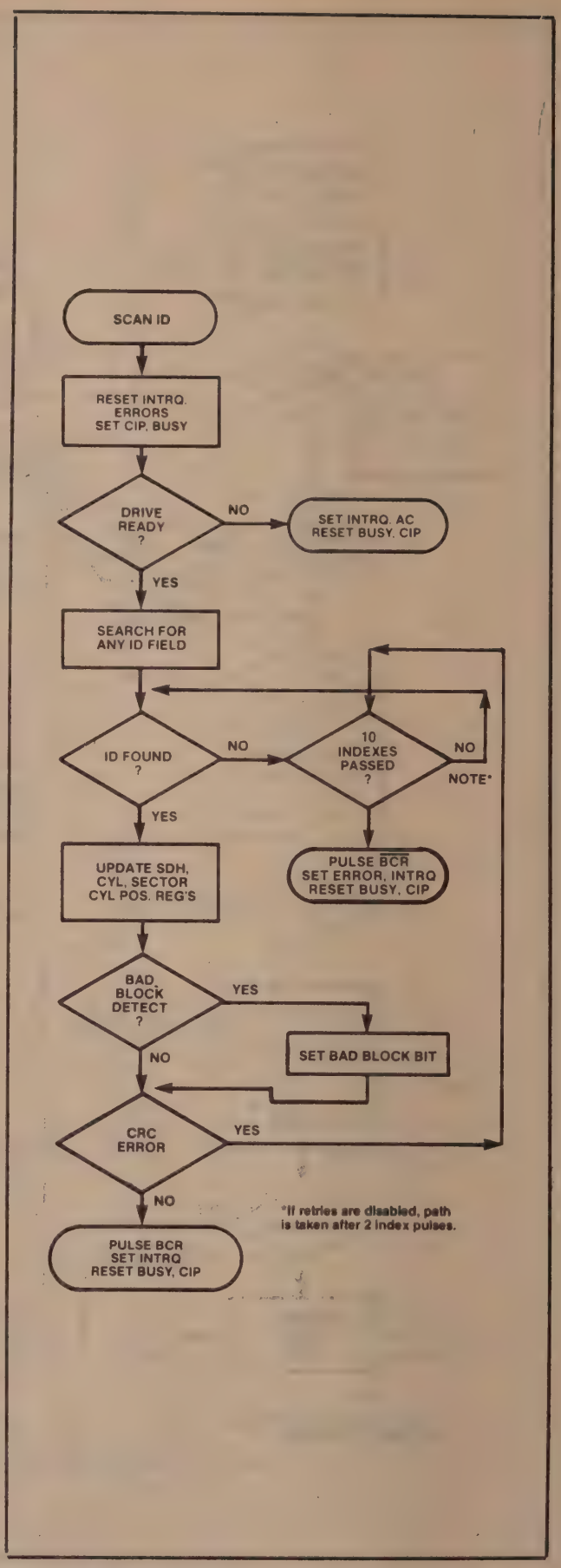


Figure 14. Scan ID Command Flow

FORMAT COMMAND SECTOR BUFFER CONTENTS

SECTOR BUFFER ADDRESS	BAD BLOCK?	LOCICAL SECTOR NUMBER
00	00	00
02	00	10
04	00	01
06	00	11
08	00	02
0A	00	12
0C	00	03
0E	00	13
10	80	04
12	00	14
14	00	05
16	00	15
18	00	06
1A	00	16
1C	00	07
1E	00	17
20	00	08
22	00	18
24	00	09
26	00	19
28	00	0A
2A	00	1A
2C	00	0B
2E	00	1B
30	00	0C
32	00	1C
34	00	0D
36	00	1D
38	00	0E
3A	00	1E
3C	00	0F
3E	00	1F
40	FF	FF
.	.	.
.	.	.
.	.	.
F0	FF	FF

Figure 15

The Gap 3 value is determined by the drive motor speed variation, data sector length, and the interleave factor. The interleave factor is only important when 1:1 interleave is used. The formula for determining the minimum Gap 3 length value is:

Gap 3 = (2 * M * S) + K + E

M = motor speed variation (e.g., 0.03 for ± 3%)

S = sector length in bytes

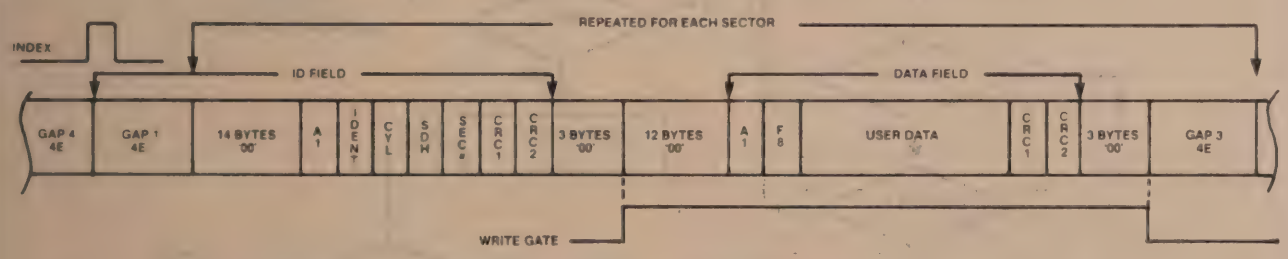
K = 25 for interleave factor of 1

K = 0 for any other interleave factor

E = 7 if the sector is to be extended

Like all commands, a WR FAULT or drive not ready condition will terminate execution of the WRITE FORMAT command. Figure 16 shows the format that the 82062 will write on the disk.

A flowchart of the WRITE FORMAT command is shown in Figure 17.



ID FIELD
A1 = A1H with 0AH clock
IDENT = MSB of Cylinder Number
FE = 0-255 Cylinders
FF = 256-511 Cylinders
FC = 512-767 Cylinders
FD = 768-1023 Cylinders

SDH BYTE = Bits 1, 1, 2 = Head Number
Bits 3, 4 = 0
Bits 5, 6 = Sector Size
Bit 7 = Bad Block Mark

Sec # = Logical Sector Number

DATA FIELD
A1 = A1H with 0AH clock
F8 = Data Address Mark, Normal clock
USER = Data Field 128 to 1024 Bytes

NOTES
1. GAP1 and 3 length determined by sector number register contents during formatting
2. If EXT bit in SDH register is set to 1 then an additional 7 data bytes are written no CRC bytes are written

Figure 16. Track Format

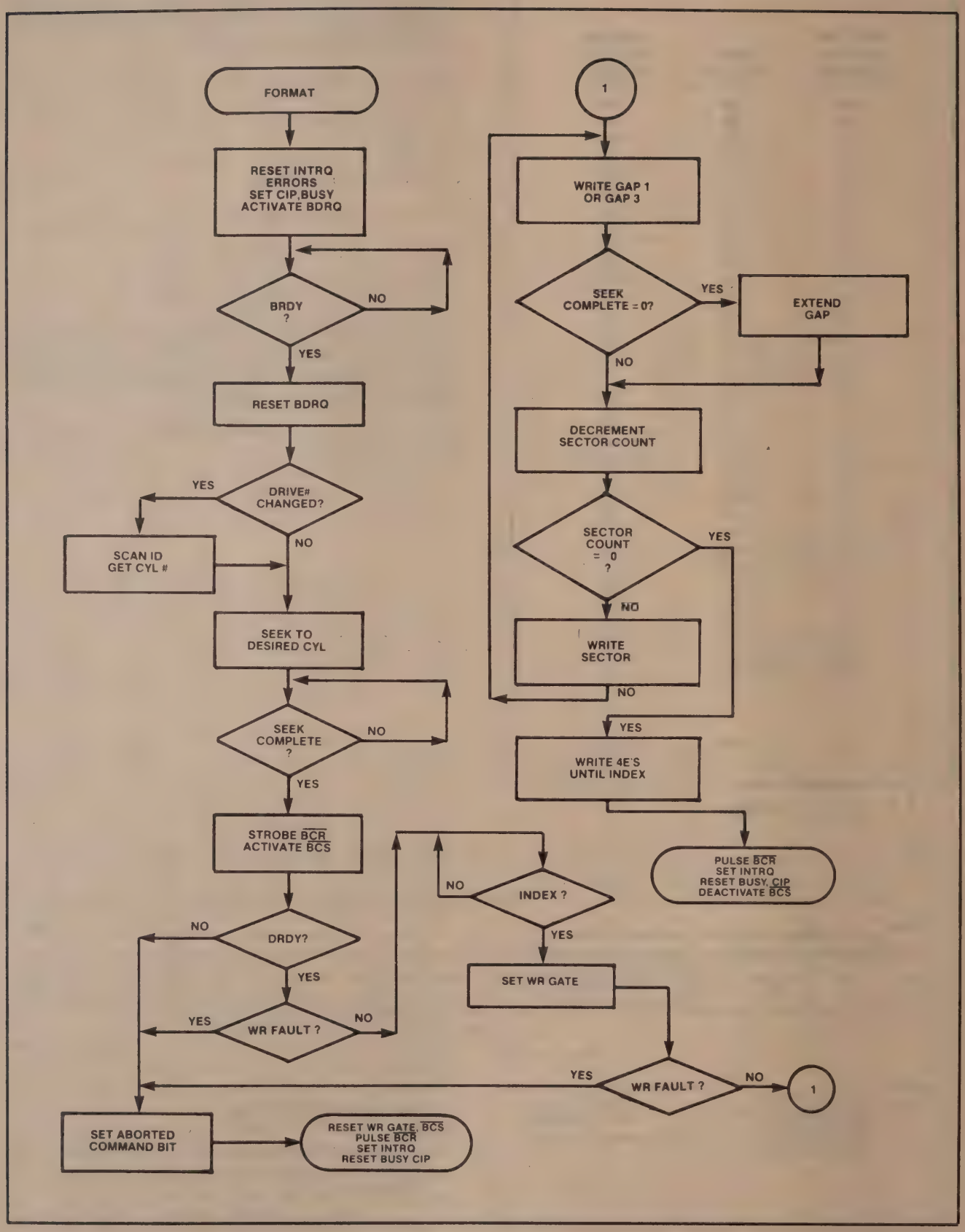


Figure 17. Write Format Command Flow

ELECTRICAL CHARACTERISTICS
ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias ... 0°C to 70°C
Storage Temperature ..... -65°C to +150°C
Voltage on any pin with
respect to GND ..... -0.5V to +7V
Power Dissipation .....1.5 Watt

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

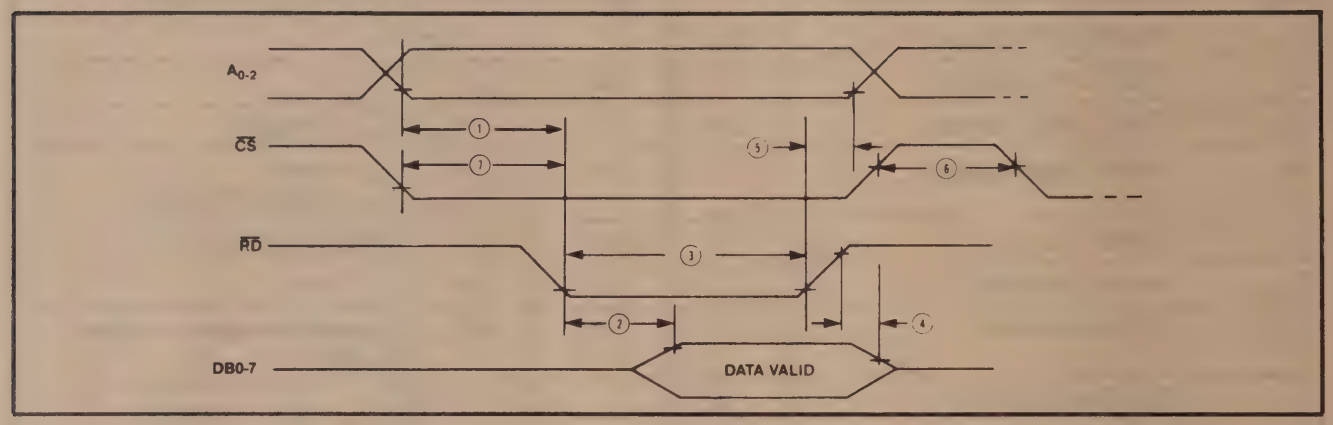
D.C. CHARACTERISTICS (T_A = 0°C to 70°C; V_{CC} = +5V ± 10%; GND = 0V)

SYMBOL	PARAMETER	MIN	MAX	UNIT	TEST CONDITIONS
I_{IL}	Input Leakage Current		±10	μA	$V_{IN} = V_{CC}$ to 0V
I_{OFL}	Output Leakage Current		±10	μA	$V_{OUT} = V_{CC}$ to 0.45V
V_{IH}	Input High Voltage	2.0		V	
V_{IL}	Input Low Voltage		0.8	V	
V_{OH}	Output High Voltage	2.4		V	I_{OH} = -100uA
V_{OL}	Output Low Voltage		0.45	V	I_{OL} = 1.6mA 4.8mA P21,22,23
I_{CC}	Supply Current		200	mA	All Outputs Open
C_{IN}	Input Capacitance		10	pF	f_c = 1 MHz
$C_{I/O}$	I/O Capacitance		20	pF	Unmeasured pins returned to GND
	For Pins 25,34,37,39				
V_{IH}	Input High Voltage	4.6		V	
V_{IL}	Input Low Voltage		0.5	V	
TRS	Rise Time		30	ns	10% to 90% points

A.C. CHARACTERISTICS (T_A = 0°C to 70°C; V_{CC} = +5V ± 10%; GND = 0V)

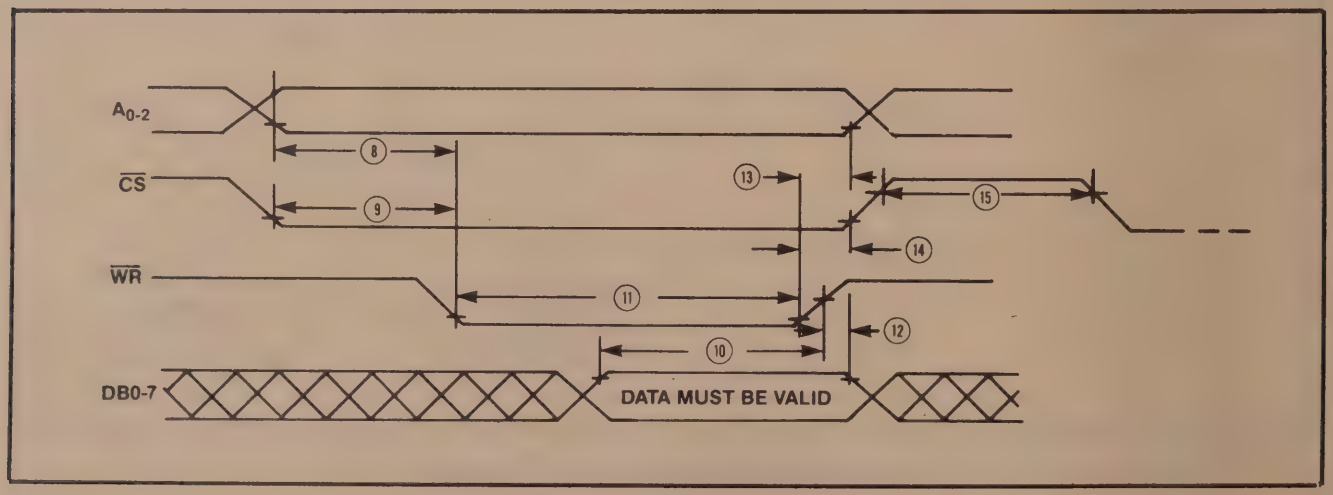
HOST READ TIMING

SYMBOL	PARAMETER	MIN	MAX	UNIT	TEST CONDITIONS
1	Address Stable Before $\overline{RD}$↓	100		ns	
2	Data Delay From $\overline{RD}$↓		375	ns	
3	$\overline{RD}$ Pulse Width	0.4	10	μs	
4	$\overline{RD}$ to Data Floating	20	200	ns	
5	Address Hold Time after $\overline{RD}$↑	0		ns	
6	Read Recovery Time	300		ns	
7	$\overline{CS}$ Stable before $\overline{RD}$↓	0		ns	See Note 6



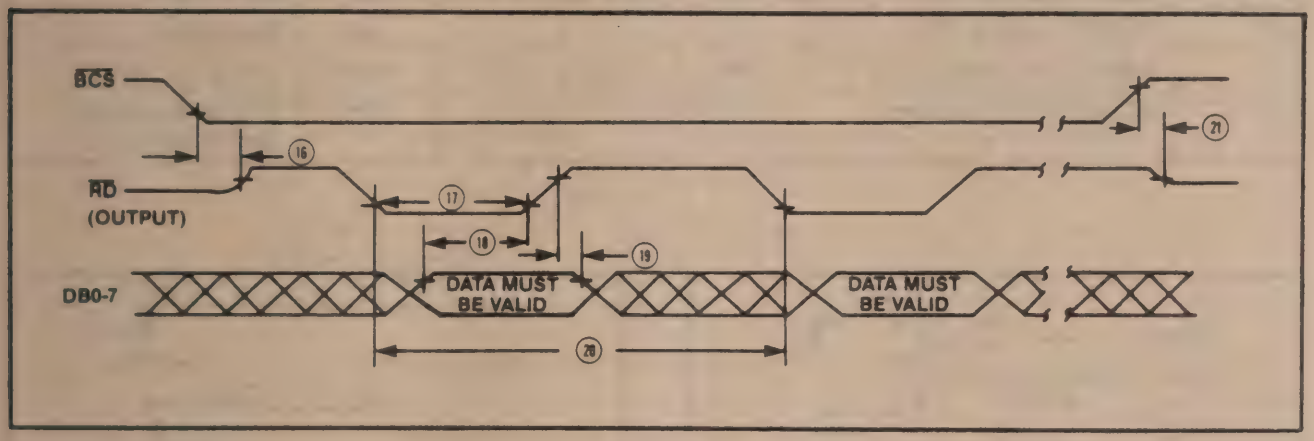
HOST WRITE TIMING

SYMBOL	PARAMETER	MIN	MAX	UNIT	TEST CONDITIONS
8	Address Stable Before $\overline{WR}$↓	0	10	μs	
9	$\overline{CS}$ Stable Before $\overline{WR}$↓	0	10	μs	
10	Data Setup Time Before $\overline{WR}$↑	0.2	10	μs	
11	$\overline{WR}$ Pulse Width	0.2	10	μs	
12	Data Hold Time After $\overline{WR}$↑	10		ns	
13	Address Hold Time After $\overline{WR}$↑	30		ns	
14	$\overline{CS}$ Hold Time After $\overline{WR}$↑	0		ns	See Note 7
15	Write Recovery Time	1.0		μs	



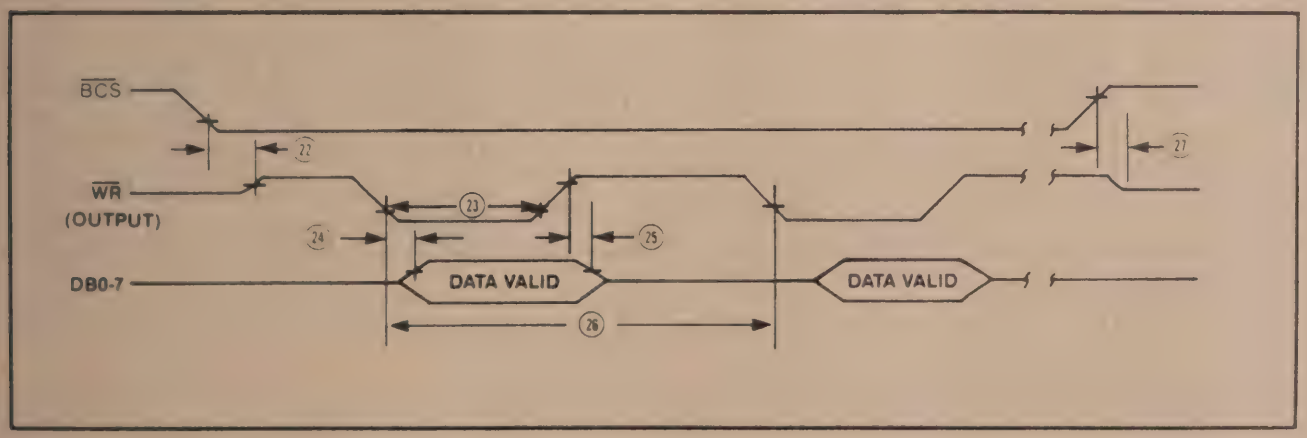
BUFFER READ TIMING (WRITE SECTOR COMMAND)

SYMBOL	PARAMETER	MIN	TYP	MAX	UNIT	TEST CONDITIONS
16	$\overline{BCS}$↓ to $\overline{RD}$ Valid	15		100	ns	
17	$\overline{RD}$ Output Pulse Width	300	400	500	ns	See Note 3
18	Data Setup to $\overline{RD}$↑	140			ns	
19	Data Hold from $\overline{RD}$↑	0			ns	
20	$\overline{RD}$ Repetition Rate	1.2	1.6	2.0	μs	See Note 1
21	$\overline{RD}$ Float from $\overline{BCS}$↑	15		100	ns	



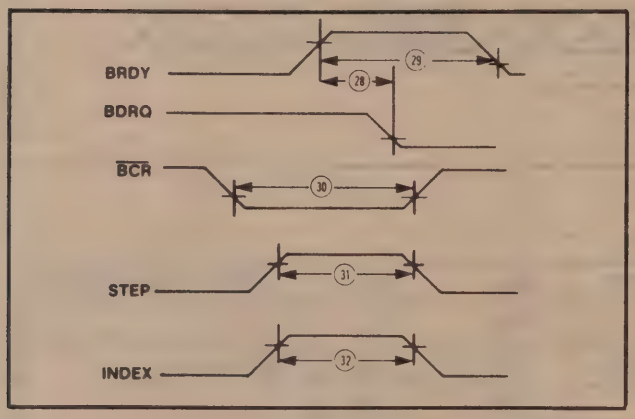
BUFFER WRITE TIMING (READ SECTOR COMMAND)

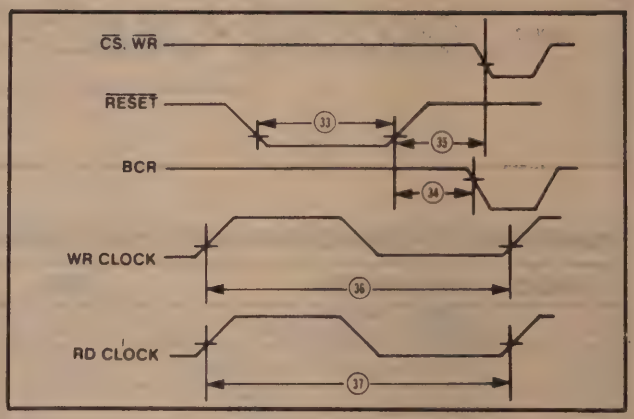
SYMBOL	PARAMETER	MIN	TYP	MAX	UNIT	TEST CONDITIONS
22	$\overline{BCS}$↓ to $\overline{WR}$ Valid	15		100	ns	
23	$\overline{WR}$ Output Pulse Width	300	400	500	ns	See Note 3
24	Data Valid from $\overline{WR}$↓			150	ns	
25	Data Hold from $\overline{WR}$↑	60			ns	
26	$\overline{WR}$ Repetition Rate	1.2	1.6	2.0	μs	See Note 1
27	$\overline{WR}$ Float from $\overline{BCS}$↑	15		100	ns	



MISCELLANEOUS TIMING

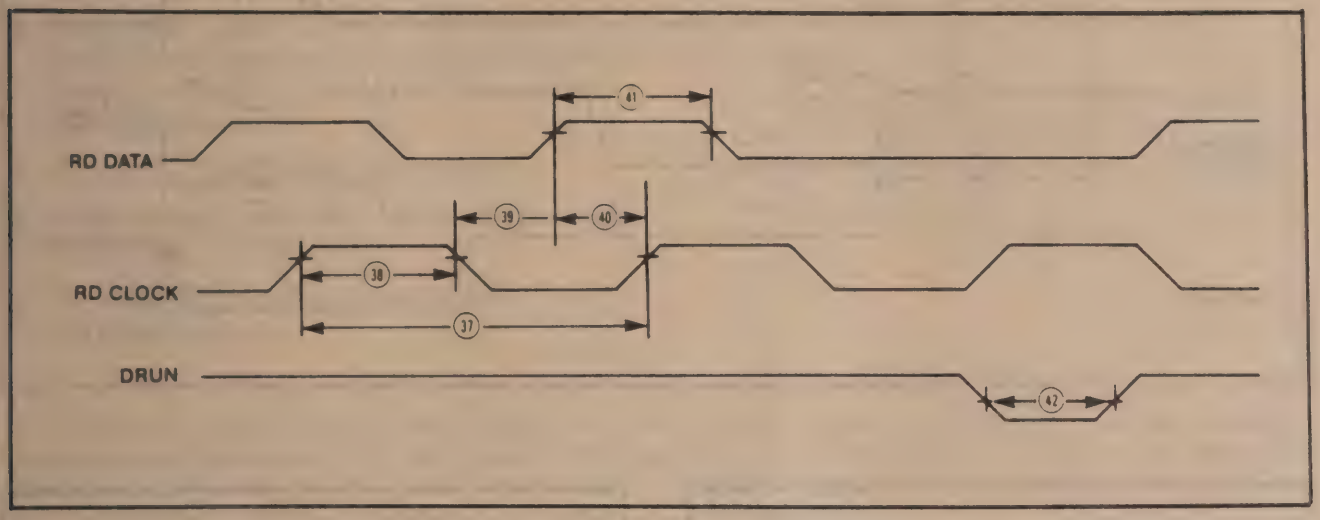
SYMBOL	PARAMETER	MIN	TYP	MAX	UNIT	TEST CONDITIONS
28	BDRQ Reset from BRDY	40		200	ns	
29	BRDY Pulse Width	800			ns	See Note 4
30	$\overline{BCR}$ Pulse Width	1.4	1.6	1.8	μs	See Note 1
31	STEP Pulse Width	8.3	8.4	8.7	μs	See Note 1
32	INDEX Pulse Width	500			ns	
33	$\overline{RESET}$ Pulse Width	24			WR CLK	See Note 2
34	$\overline{RESET}$↑ to $\overline{BCR}$	1.6	3.2	6.4	μs	See Note 1
35	$\overline{RESET}$↑ to $\overline{WR}$, $\overline{CS}$↓	6.4			μs	See Note 1
36	WR CLOCK Frequency	0.25	5.0	5.25	MHz	50% Duty Cycle
37	RD CLOCK Frequency	0.25	5.0	5.25	MHz	50% Duty Cycle See Note 5





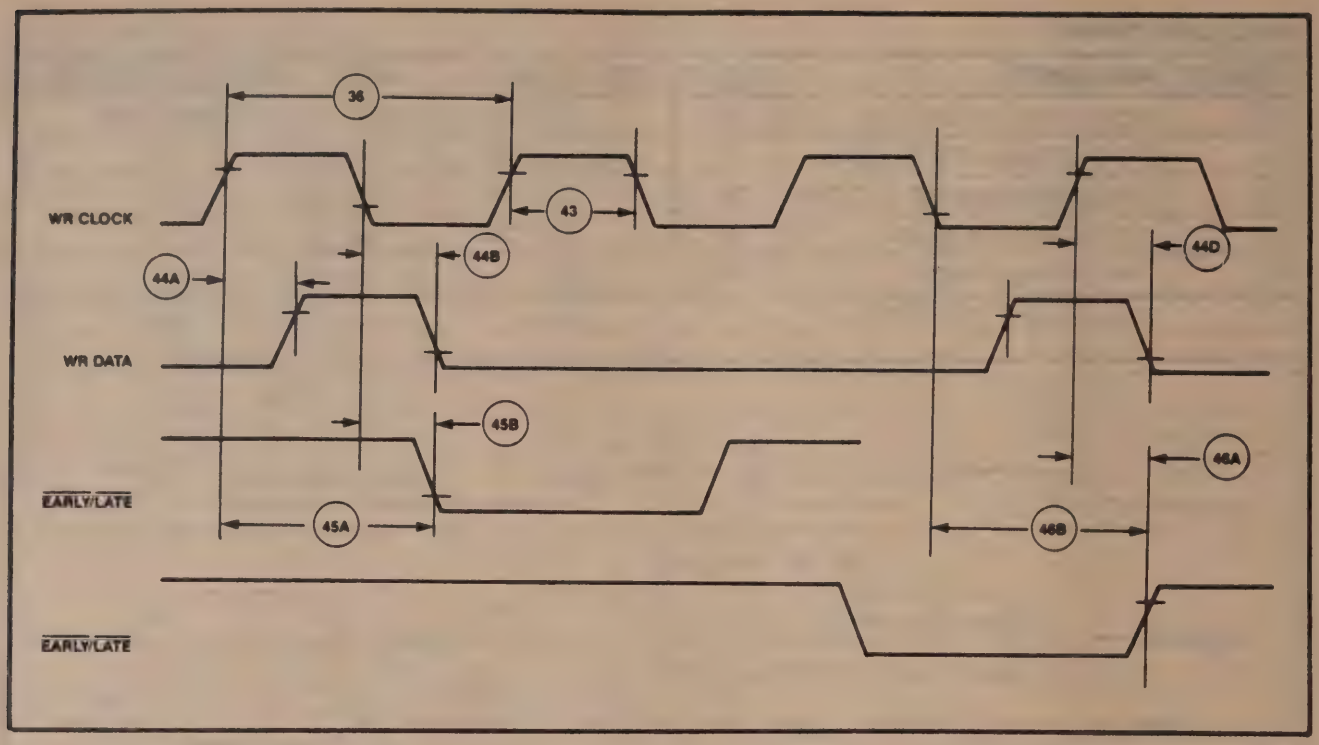
READ DATA TIMING

SYMBOL	PARAMETER	MIN	TYP	MAX	UNIT	TEST CONDITIONS
38	RD CLOCK Pulse Width	95		2000	ns	50% Duty Cycle
39	RD DATA after RD CLOCK↓	0		T38	ns	
40	RD DATA before RD CLOCK↑	20		T38	ns	
41	RD DATA Pulse Width	40		T38	ns	
42	DRUN Pulse Width	30			ns	

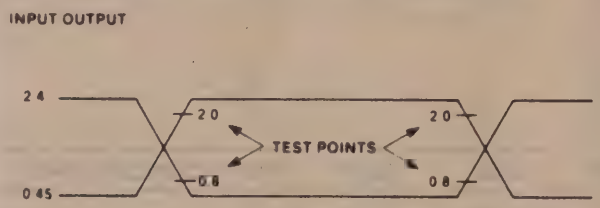


WRITE DATA TIMING

SYMBOL	PARAMETER	MIN	TYP	MAX	UNIT	TEST CONDITIONS
43	WR CLOCK Pulse Width	95		2000	ns	
	Propogation Delay					
44A	WR CLOCK↑ to WR DATA↑	10		65	ns	
44B	WR CLOCK↓ to WR DATA↓					
44D	WR CLOCK↓ to WR DATA↑					
45A	WR CLOCK↑ to $\overline{\text{EARLY}}$/$\overline{\text{LATE}}$↓	10		65	ns	
45B	WR CLOCK↓ to $\overline{\text{EARLY}}$/$\overline{\text{LATE}}$↓					
46A	WR CLOCK↑ to $\overline{\text{EARLY}}$/$\overline{\text{LATE}}$↑	10		65	ns	
46B	WR CLOCK↓ to $\overline{\text{EARLY}}$/$\overline{\text{LATE}}$↑					

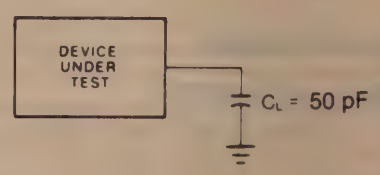


A.C. TESTING INPUT, OUTPUT WAVEFORM



AC TESTING: INPUTS ARE DRIVEN AT 2.4V FOR A LOGIC .1, AND 0.45V FOR A LOGIC .0. TIMING MEASUREMENTS ARE MADE AT 2.0V FOR A LOGIC .1, AND0.8V FOR A LOGIC .0.

A.C. TESTING LOAD CIRCUIT



C_L INCLUDES JIG CAPACITANCE

NOTES:

1. Based on WR CLOCK = 5.0 MHz.
2. 24 WR CLOCK periods = 4.8 μs at 5.0 MHz.
3. 2 WR CLOCK periods ± 100 ns.
4. When used with a DMA controller BRDY must be > 4 μs or a spurious BDRQ pulse may exist for up to 4 μs after the rising edge of BRDY.
5. WR CLOCK Frequency = RD CLOCK Frequency ± 15%.
6. $\overline{RD}$ may be asserted before $\overline{CS}$ as long as it remains active for at least the minimum. T3 pulse width after $\overline{CS}$ is asserted.
7. $\overline{WR}$ may be asserted before $\overline{CS}$ as long as it remains active for at least the minimum T11 pulse width after $\overline{CS}$ is asserted.

PRELIMINARY

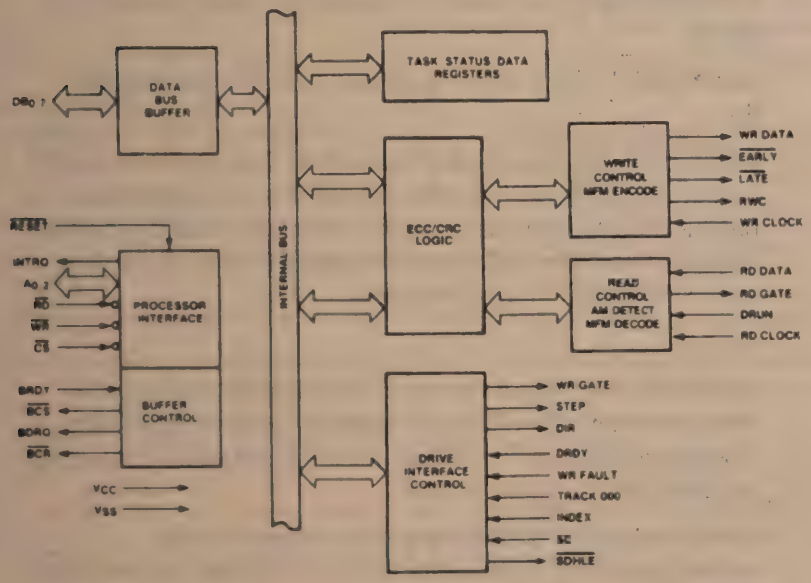
82064
WINCHESTER DISK CONTROLLER WITH ON-CHIP ERROR DETECTION AND CORRECTION

- Compatible with all Intel and most other microprocessors
- Controls ST506/ST412 Interface Winchester Disk Drives
- 5 Mbit/sec Data Transfer Rate
- Eight High-Level commands: Restore, Seek, Read Sector, Write Sector, Scan ID, Write Format, Compute Correction, Set Parameter
- Software Compatible with 82062
- High-speed "zero wait state" operation with 8 MHz 80186/188
- On-chip ECC Unit Automatically corrects errors
- 5 or 11-bit correction - span software selectable
- Implied seeks with Read/Write Commands
- Multiple Sector Transfer Capability
- 128, 256, 512 and 1024 Byte Sector Lengths
- Available in 40-Lead Ceramic Dual In-Line, 40-Lead Plastic Dual In-Line, and 44-Lead Plastic Chip Carrier Packages

(See Packaging Spec., Order #231369)

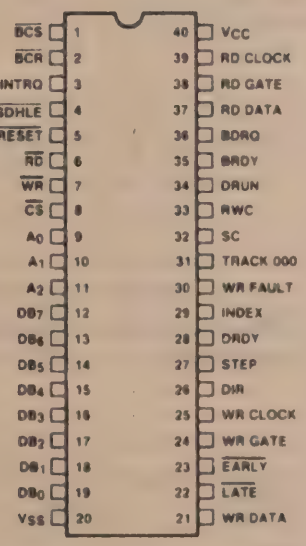
The 82064 Winchester Disk Controller (WDC) with on-chip error detection and correction circuitry interfaces microprocessor systems to 5¼" Winchester disk drives. It is socket and software compatible with the 82062 Winchester Disk Controller, and additionally includes on-chip ECC, support for drives with up to 2k tracks, and has an additional control signal which eliminates an external decoder.

The 82064 is fabricated on Intel's advanced HMOS III technology and is available in 40-pin CERDIP and plastic packages.



231242-1

Figure 1. 82064 Block Diagram



231242-2

Figure 2. 82064 Pinout

Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied. Information contained herein supersedes previously published specifications on these devices from Intel. November 1985
© Intel Corporation, 1985
Order Number: 231242-002

Table 1. Pin Description

Symbol	Pin No.	Type	Name and Function
$\overline{BCS}$	1	O	**BUFFER CHIP SELECT:** Output used to enable reading or writing of the external sector buffer by the 82064. When low, the host should not be able to drive the 82064 data bus, $\overline{RD}$, or $\overline{WR}$ lines.
$\overline{BCR}$	2	O	**BUFFER COUNTER RESET:** Output that is asserted by the 82064 prior to read/write operation. This pin is asserted whenever $\overline{BCS}$ changes state. Used to reset the address counter of the buffer memory.
INTRQ	3	O	**INTERRUPT REQUEST:** Interrupt generated by the 82064 upon command termination. It is reset when the STATUS register is read, or a new command is written to the COMMAND register. Optionally signifies when a data transfer is required on Read Sector commands.
$\overline{SDHLE}$	4	O	$\overline{SDHLE}$ is asserted when the SDH register is written by the host.
$\overline{RESET}$	5	I	**RESET:** Initializes the controller and clears all status flags. Does not clear the Task Register File.
$\overline{RD}$	6	I/O	**READ:** Tri-state, bi-directional signal. As an input, $\overline{RD}$ controls the transfer of information from the 82064 registers to the host. $\overline{RD}$ is an output when the 82064 is reading data from the sector buffer ($\overline{BCS}$ low).
$\overline{WR}$	7	I/O	**WRITE:** Tri-state, bi-directional signal. As an input, $\overline{WR}$ controls the transfer of command or task information into the 82064 registers. $\overline{WR}$ is an output when the 82064 is writing data to the sector buffer ($\overline{BCS}$ low).
$\overline{CS}$	8	I	**CHIP SELECT:** Enables $\overline{RD}$ and $\overline{WR}$ as inputs for access to the Task Registers. It has no effect once a disk command starts.
A_{0-2}	9–11	I	**ADDRESS:** Used to select a register from the task register file.
DB_{0-7}	12–19	I/O	**DATA BUS:** Tri-state, bi-directional 8-bit Data Bus with control determined by BCS. When BCS is high the microprocessor has full control of the data bus for reading and writing the Task Register File. When BCS is low the 82064 controls the data bus to transfer to or from the buffer.
V_{ss}	20		Ground
WR DATA	21	O	**WRITE DATA:** Output that shifts out MFM data at a rate determined by Write Clock. Requires an external D flip-flop clocked at 10 MHz. The output has an active pullup and pulldown that can sink 4.8 mA.
$\overline{LATE}$	22	O	**LATE:** Output used to derive a delay value for write precompensation. Valid when WR GATE is high. Active on all cylinders.
$\overline{EARLY}$	23	O	**EARLY:** Output used to derive a delay value for write precompensation. Valid when WR GATE is high. Active on all cylinders.

Table 1. Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
WR GATE	24	O	**WRITE GATE:** High when write data is valid. WR GATE goes low if the WR FAULT input is active. This output is used by the drive to enable head write current.
WR CLOCK	25	I	**WRITE CLOCK:** Clock input used to derive the write data rate. Frequency = 5 MHz for the ST506 interface.
DIR	26	O	**DIRECTION:** High level on this output tells the drive to move the head inward (increasing cylinder number). The state of this signal is determined by the 82064's internal comparison of actual cylinder location vs. desired cylinder.
STEP	27	O	**STEP:** This signal is used to move the drive head to another cylinder at a programmable frequency. Pulse width = 1.6 μs for a step rate of 3.2 μs/step, and 8.4 μs for all other step rates.
DRDY	28	I	**DRIVE READY:** If DRDY from the drive goes low, the command will be terminated.
INDEX	29	I	**INDEX:** Signal from the drive indicating the beginning of a track. It is used by the 82064 during formatting, and for counting retries. Index is edge triggered. Only the rising edge is valid.
WR FAULT	30	I	**WRITE FAULT:** An error input to the 82064 which indicates a fault condition at the drive. If WR FAULT from the drive goes high, the command will be terminated.
TRACK 000	31	I	**TRACK ZERO:** Signal from the drive which indicates that the head is at the outermost cylinder. Used to verify proper completion of a RESTORE command.
SC	32	I	**SEEK COMPLETE:** Signal from the drive indicating to the 82064 that the drive head has settled and that reads or writes can be made. SC is edge triggered. Only the rising edge is valid.
RWC	33	O	**REDUCED WRITE CURRENT:** Signal goes high for all cylinder numbers above the value programmed in the Write Precomp Cylinder register. It is used by the precompensation logic and by the drive to reduce the effects of bit shifting.
DRUN	34	I	**DATA RUN:** This signal informs the 82064 when a field of all ones or all zeroes has been detected in the read data stream by an external one-shot. This indicates the beginning of an ID field. RD GATE is brought high when DRUN is sampled high for 16 clock periods.
BRDY	35	I	**BUFFER READY:** Input used to signal the controller that the buffer is ready for reading (full), or writing (empty), by the host μP. Only the rising edge indicates the condition.

Table 1. Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
BDRQ	36	O	**BUFFER DATA REQUEST:** Activated during Read or Write commands when a data transfer between the host and the 82064's sector buffer is required. Typically used as a DMA request line.
RD DATA	37	I	**READ DATA:** Single ended input that accepts MFM data from the drive.
RD GATE	38	O	**READ GATE:** Output that is asserted when a search for an address mark is initiated. It remains asserted until the end of the ID or data field.
RD CLOCK	39	I	**READ CLOCK:** Clock input derived from the external data recovery circuits.
V_{CC}	40	I	**D.C. POWER:** +5V.

FUNCTIONAL DESCRIPTION

The Intel 82064 Winchester Disk Controller (WDC) interfaces microprocessor systems to Winchester disk drives that use the Seagate Technology ST506/ST412 interface. The device translates parallel data from the microprocessor to a 5 Mbit/sec, MFM-encoded serial bit stream. It provides all of the drive control logic and control signals which simplify the design of external data separation and write precompensation circuitry. The 82064 is designed to interface to the host processor through an external sector buffer.

On-chip error detection algorithms include the CRC/CCITT and a 32-bit computer generated ECC polynomial. If the ECC code is selected, the 82064 provides three possible error handling techniques if an error is detected during a read operation:

1. Automatically correct the data in the sector buffer, providing the host with good information.
2. Provide the host with the error location and pattern, allowing the host to correct the error.
3. Take no action other than setting the error flag.

The 82064 is software compatible with the 82062.

INTERNAL ARCHITECTURE

The internal architecture of the 82064 is shown in more detail in Figure 3. It is made up of seven major blocks as described below.

PLA Controller

The PLA interprets commands and provides all control functions. It is synchronized with WR CLOCK.

Magnitude Comparator

An 11-bit magnitude comparator is used to calculate the direction and number of steps needed to move the heads from the present to the desired cylinder position. It compares the cylinder number in the task file to the internal "present position" cylinder number.

A separate high-speed equivelance comparator is used to compare ID field bytes when searching for a sector ID field.

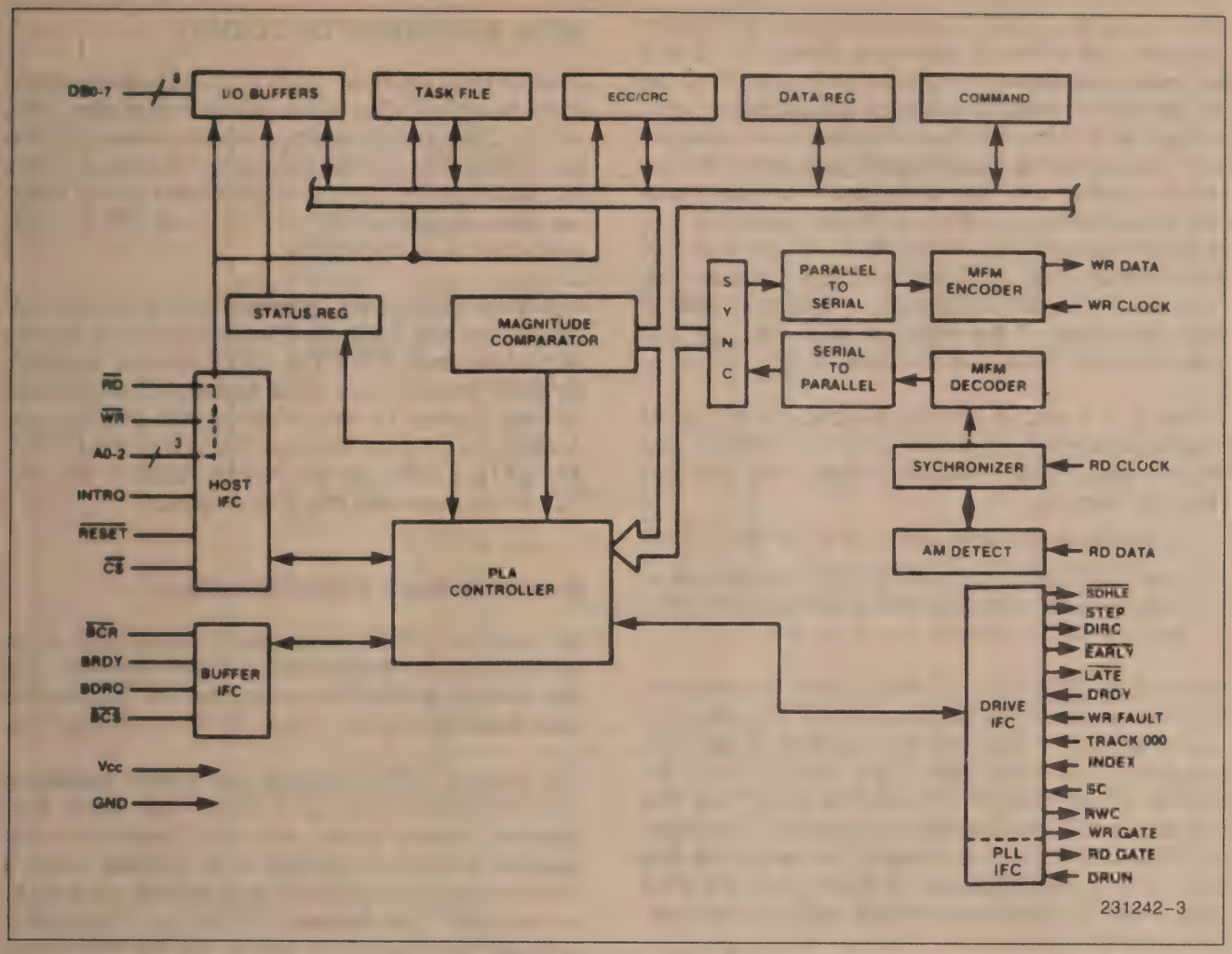


Figure 3. 82064 Detailed Block Diagram

CRC and ECC Generator and Checker

The 82064 provides two options for protecting the integrity of the data field. The data field may have either a CRC (SDH register, bit 7=0), or a 32-bit ECC (SDH register, bit 7=1) appended to it. The ID field is always protected by a CRC.

The CRC mode provides a means of verifying the accuracy of the data read from the disk, but does not attempt to correct it. The CRC generator computes and checks cyclic redundancy check characters that are written and read from the disk after ID and data fields. The polynomial used is:

$X^{16}+X^{12}+X^{5}+1$

The CRC register is preset to all one's before computation starts.

If the CRC character generated while reading the data does not equal the one previously written an error exists. If an ID field CRC error occurs the "ID not found" bit in the error register will be set. If a data field CRC error occurs the "ECC/CRC" bit in the error register will be set.

The ECC mode is only applicable to the data field. It provides the user with the ability to detect and correct errors in the data field automatically. The commands and registers which must be considered when ECC is used are:

1. SDH Register, bit 7 (CRC/ECC)
2. READ SECTOR Command, bit 0 (T)
3. READ SECTOR and WRITE SECTOR Commands, bit 1 (L)
4. COMPUTE CORRECTION Command
5. SET PARAMETER Command
6. STATUS Register, bit 2 - error correction successful
7. STATUS Register, bit 0 - error occurred
8. ERROR Register, bit 6 - uncorrectable error

To enable the ECC mode, bit 7 of the SDH register must be set to one.

Bit 0 (T) of the READ Command controls whether or not error correction is attempted. When T = 0 and an error is detected, the 82064 tries up to 10 times to correct the error. If the error is successfully corrected, bit 2 of the STATUS Register is set. The host can interrogate the status register and detect that an error occurred and was corrected. If the error was not correctable, bit 6 of the ERROR Register is set. If the correction span was set to 5 bits, the host may now execute the SET PARAMETER Command to change the correction span to 11 bits, and attempt the read again. If the error persists, the host can read the data, but it will contain errors.

When T = 1 and an error is detected, no attempt is made to correct it. Bit 0 of the STATUS Register and bit 6 of the ERROR Register are set. The user now has two choices:

1. Ignore the error and make no attempt to correct it.
2. Use the COMPUTE CORRECTION Command to determine the location and pattern of the error, and correct it within the user's program.

When the COMPUTE CORRECTION Command is implemented, it must be done before executing any command which can alter the contents of the ECC Register. The READ SECTOR, WRITE SECTOR, SCAN ID, and FORMAT Commands will alter this register and correction will be impossible. The COMPUTE CORRECTION Command may determine that the error is uncorrectable, at which point the error bits in the STATUS and ERROR Registers are set.

Although ECC generation starts with the first bit of the F8H byte in the data ID field, the actual ECC bytes written will be the same as if the A1H byte was included. The ECC polynomial used is:

$$X^{32}+X^{28}+X^{26}+X^{19}+X^{17}+X^{10}+X^{6}+X^{2}+1$$

For automatic error correction, the external sector buffer must be implemented with a static RAM and counter, not with a FIFO.

The SET PARAMETER Command is used to select a 5-bit or 11-bit correction span.

When the L Bit (bit 1) of the READ SECTOR and WRITE SECTOR commands is set to one, they are referred to as READ LONG and WRITE LONG commands. For these commands, no CRC or ECC characters are generated or checked by the 82064. In effect, the data field is extended by 4 bytes which are passed to/from the sector buffer.

With proper use of the WRITE SECTOR, READ LONG, WRITE LONG, and READ SECTOR Commands, a diagnostic routine may be developed to test the accuracy of the error correction process.

MFM ENCODER/DECODER

Encodes and decodes MFM data to be written/read from the drive. The MFM encoder operates from WR CLOCK, a clock having a frequency equal to the bit rate. The MFM decoder operates from RD CLOCK, a bit rate clock generated by the external data separator. RD CLOCK and WR CLOCK need not be synchronous.

The MFM encoder also generates the write precompensation control signals. Depending on the bit pattern of the data, $\overline{\text{EARLY}}$ or $\overline{\text{LATE}}$ may be asserted. External circuitry uses these signals to compensate for drift caused by the influence one bit has over another. More information on the use of the $\overline{\text{EARLY}}$ and $\overline{\text{LATE}}$ control signals can be found in the section which describes the drive interface.

Address Mark (AM) Detection

An address mark is comprised of two unique bytes preceeding both the ID field and the data field. The first byte is used for resynchronization. The second byte indicates whether it is an ID field or a data field.

The first byte, A1H, normally has a clock pattern of 0EH; however, one clock pulse has been suppressed, making it 0AH. With this pattern, the AM detector knows it is looking at an address mark. It now examines the next byte to determine if it is an ID or data field. If this byte is 111101XX or 111111XX it is an ID field. Bits 3, 1, and 0 are the high order cylinder number bits. If the second byte is F8H, it is a data field.

Host/Buffer Interface Control

The primary interface between the host processor and the 82064 is an 8-bit bi-directional bus. This bus is used to transmit and receive data for both the 82064 and the sector buffer. The sector buffer consists of a static RAM and counter. Since the 82064 makes the bus active when accessing the sector buffer, a transceiver must be used to isolate the host during this time. Figure 4 illustrates a typical interface with a sector buffer. Whenever the 82064 is not using the sector buffer, the BUFFER CHIP SELECT ($\overline{\text{BCS}}$) is high (disabled). This allows the host access to the 82064's Task Register File and to the sector buffer. A decoder is used to generate $\overline{\text{BCS}}$ when A_{0-2} is '000', an unused address in the 82064. A binary counter is enabled whenever $\overline{\text{RD}}$ or $\overline{\text{WR}}$ go active. The location within the sector buffer which is addressed by the counter will be accessed. The counter will be incremented by the trailing edge of the $\overline{\text{RD}}$ or $\overline{\text{WR}}$. This allows the host to access se-

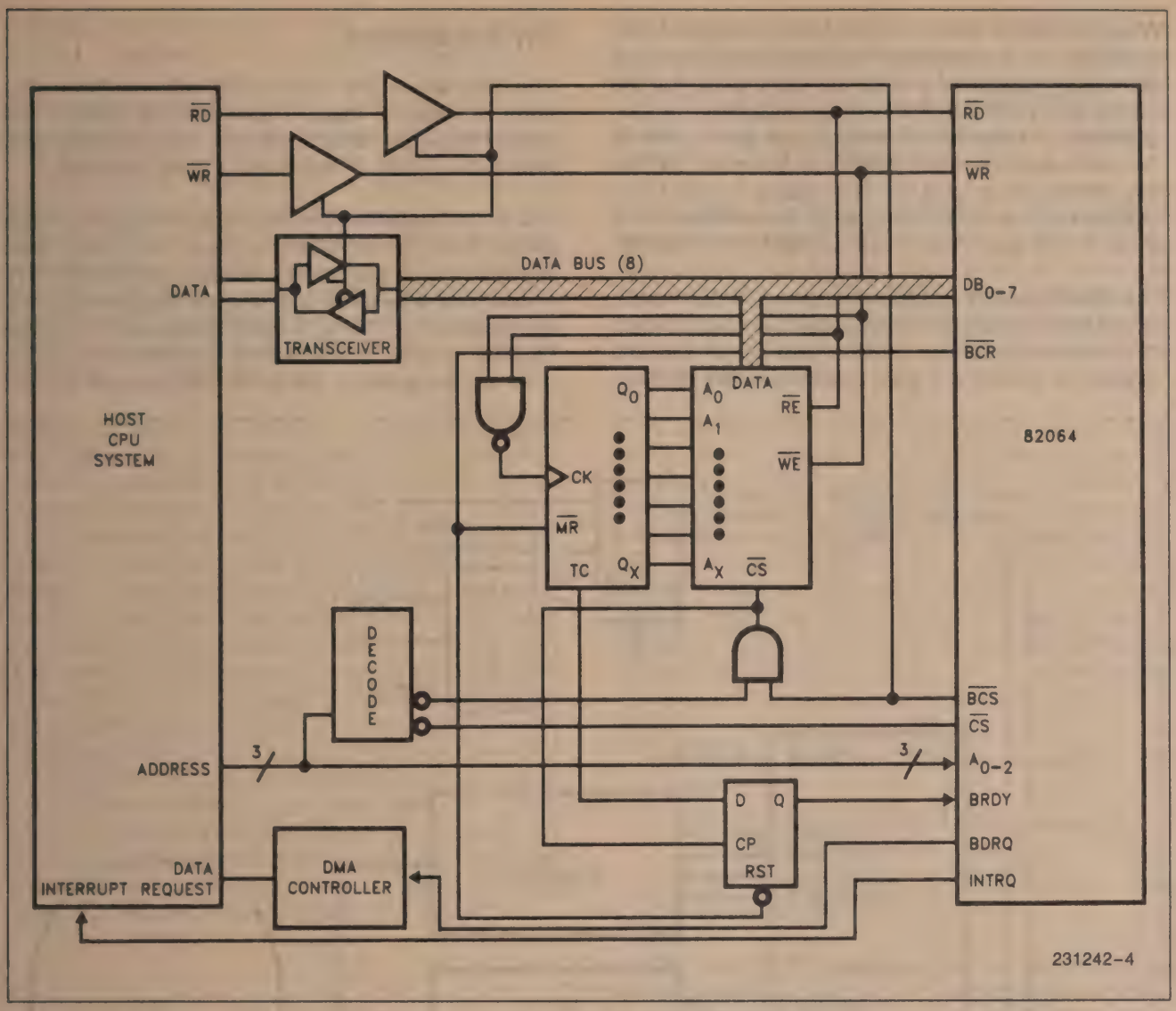


Figure 4. Host Interface Block Diagram

quential bytes within the sector buffer. The decoder also generates a $\overline{CS}$ for the 82064 whenever A_{0-2} does not equal '000', allowing access to the 82064's internal Task Register File while keeping the sector buffer tri-stated.

During a WRITE SECTOR Command, the host processor sets up data in the Task Register File and then issues the command. The 82064 asserts BUFFER COUNTER RESET ($\overline{BCR}$) to reset the counter. It then generates a status to inform the host that it can load the sector buffer with data to be written. When the counter reaches its maximum count, the BUFFER READY (BRDY) signal is asserted by the carry out of the counter, informing the 82064 that the sector buffer is full. (BRDY is a rising edge triggered signal which will be ignored if asserted before the 82064 asserts $\overline{BCR}$.) $\overline{BCS}$ is then asserted, disconnecting the host through the transceivers, and the $\overline{RD}$ and $\overline{WR}$ lines become outputs from the 82064 to allow access to the sector buffer. When the 82064 is done using the buffer, it deasserts $\overline{BCS}$ which again allows the host to access the local bus. The READ SECTOR command operates in a similar manner, except the buffer is loaded by the 82064 instead of the host.

Another control signal, BUFFER DATA REQUEST (BDRQ), can be used with a DMA controller to indicate that the 82064 is ready to send or receive data. When data transfer is via a programmed I/O environment, it is the responsibility of the host to interrogate the DRQ status bit to determine if the 82064 is ready (bit 3 of the status register). For further explanation, refer to the individual command descriptions and the A.C. Characteristics.

When INTRQ is asserted, the host is signaled that execution of a command has terminated (either a normal termination or an aborted command). For the READ SECTOR command, interrupts may be programmed to be asserted either at the termination of the command, or when BDRQ is asserted. INTRQ will remain active until the host reads the STATUS register to determine the cause of the termination, or writes a new command into the COMMAND register.

The 82064 asserts $\overline{\text{SDHLE}}$ whenever the SDH register is being written. This signal can be used to latch the drive and head select information in an external register for decoding. Figure 5 illustrates one method.

Drive Interface

The drive side of the 82064 WDC requires three sections of external logic. These are the control line buffer/receivers, data separator, and write precompensation. Figure 5 illustrates a drive interface.

The buffer/receivers condition the control lines to be driven down the cable to the drive. The control lines are typically single-ended, resistor terminated, TTL levels. The data lines to and from the drive also require buffering. This is typically done with differential RS-422 drivers. The interface specification for the drive will be found in the drive manufacturer's OEM

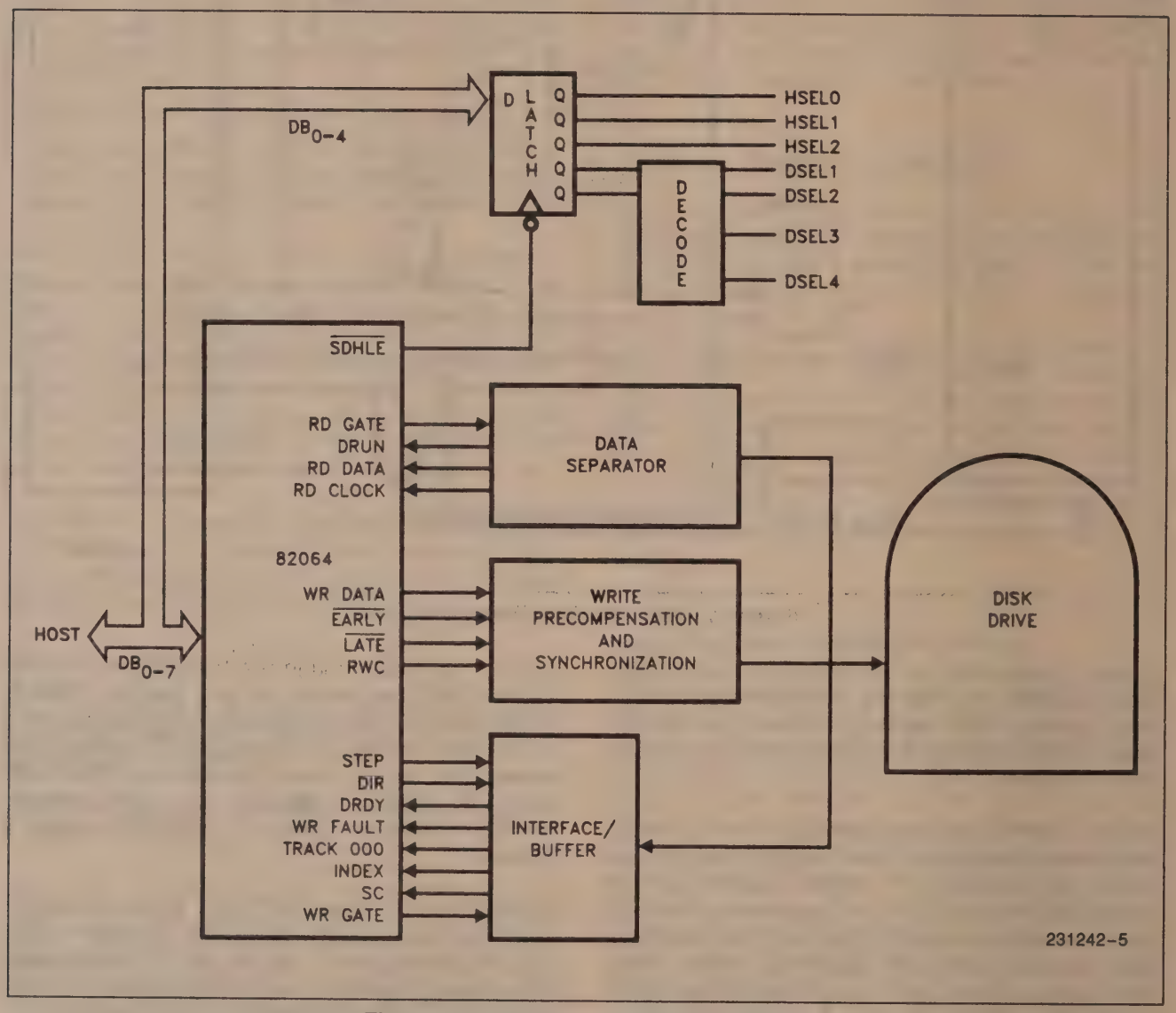


Figure 5. Drive Interface Block Diagram

manual. The 82064 supplies TTL compatible signals, and will interface to most buffer/driver devices.

The data recovery circuits consist of a phase locked loop, data separator, and associated components. The 82064 interacts with the data separator through the DATA RUN (DRUN) and RD GATE signals. A block diagram of a typical data separator circuit is shown in Figure 6. Read data from the drive is presented to the RD DATA input of the 82064, the reference multiplexor, and a retriggerable one shot. The RD GATE output will be deasserted when the 82064 is not inspecting data. The PLL should remain locked to the reference clock.

When any READ or WRITE command is initiated and a search for an address mark begins, the DRUN input is examined. The DRUN one-shot is set for slightly longer than one bit time, allowing it to retrigger constantly on a field of all ones or all zeroes. An internal counter times out to see that DRUN is asserted for two byte times. RD GATE is asserted by the 82064, switching the data separator to lock on to the incoming data stream. If DRUN is deasserted prior to an additional seven byte times, RD GATE is deasserted and the process is repeated. RD GATE will remain asserted until a non-zero, non-address mark byte is detected. The 82064 will then deassert RD GATE for two byte times to allow the PLL to lock back on the reference clock, and start the DRUN search again. If an address mark is detected, RD GATE remains asserted and the command will continue searching for the proper ID field. This sequence is shown in the flow chart in Figure 7.

The write precompensation circuitry is designed to reduce the drift in the data caused by interaction between bits. It is divided into two parts, REDUCED WRITE CURRENT (RWC) and $\overline{\text{EARLY}}$/$\overline{\text{LATE}}$ writing of bits. A block diagram of a typical write precompensation circuit is shown in Figure 8.

The cylinder in which the RWC line becomes active is controlled by the REDUCE WRITE CURRENT register in the Task Register File. When a cylinder is written which has a cylinder number greater than or equal to the contents of this register, the write current will be reduced. This will decrease the interaction between the bits.

Drift may also be caused by the bit pattern. With certain combinations of ones and zeroes some of the bits can drift far enough apart to be difficult to read without error. This phenomenon can be minimized by using $\overline{\text{EARLY}}$ and $\overline{\text{LATE}}$ as described below. The 82064 examines three bits, the last one written, the one being written, and the next one to be written. From this, it determines whether to assert $\overline{\text{EARLY}}$ or $\overline{\text{LATE}}$. Since the bit leaving the 82064 has already been written, it is too late to make it early. Therefore, the external delay circuit must be as follows:

$\overline{\text{EARLY}}$ asserted and $\overline{\text{LATE}}$ deasserted = no delay

$\overline{\text{EARLY}}$ deasserted and $\overline{\text{LATE}}$ deasserted = one unit delay (typically 12–15 ns)

$\overline{\text{EARLY}}$ deasserted and $\overline{\text{LATE}}$ asserted = two units delay (typically 24–30 ns)

$\overline{\text{EARLY}}$ and $\overline{\text{LATE}}$ are always active, and should be gated externally by the RWC signal. Figure 8 illustrates one method of using these signals.

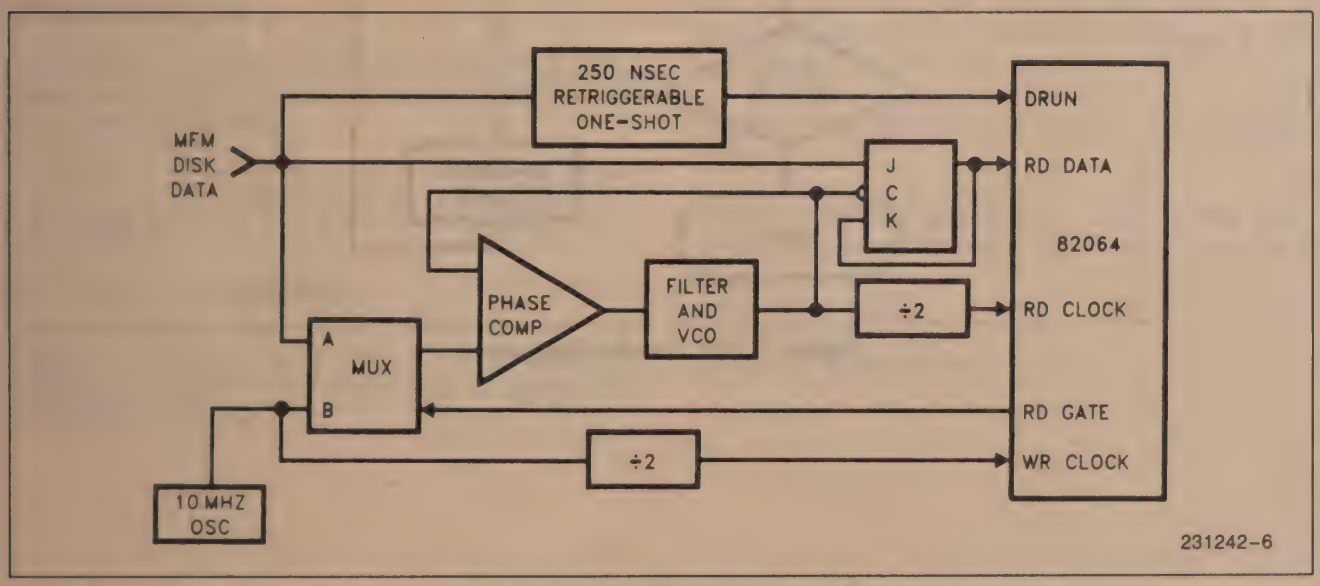


Figure 6. Data Separator Circuit

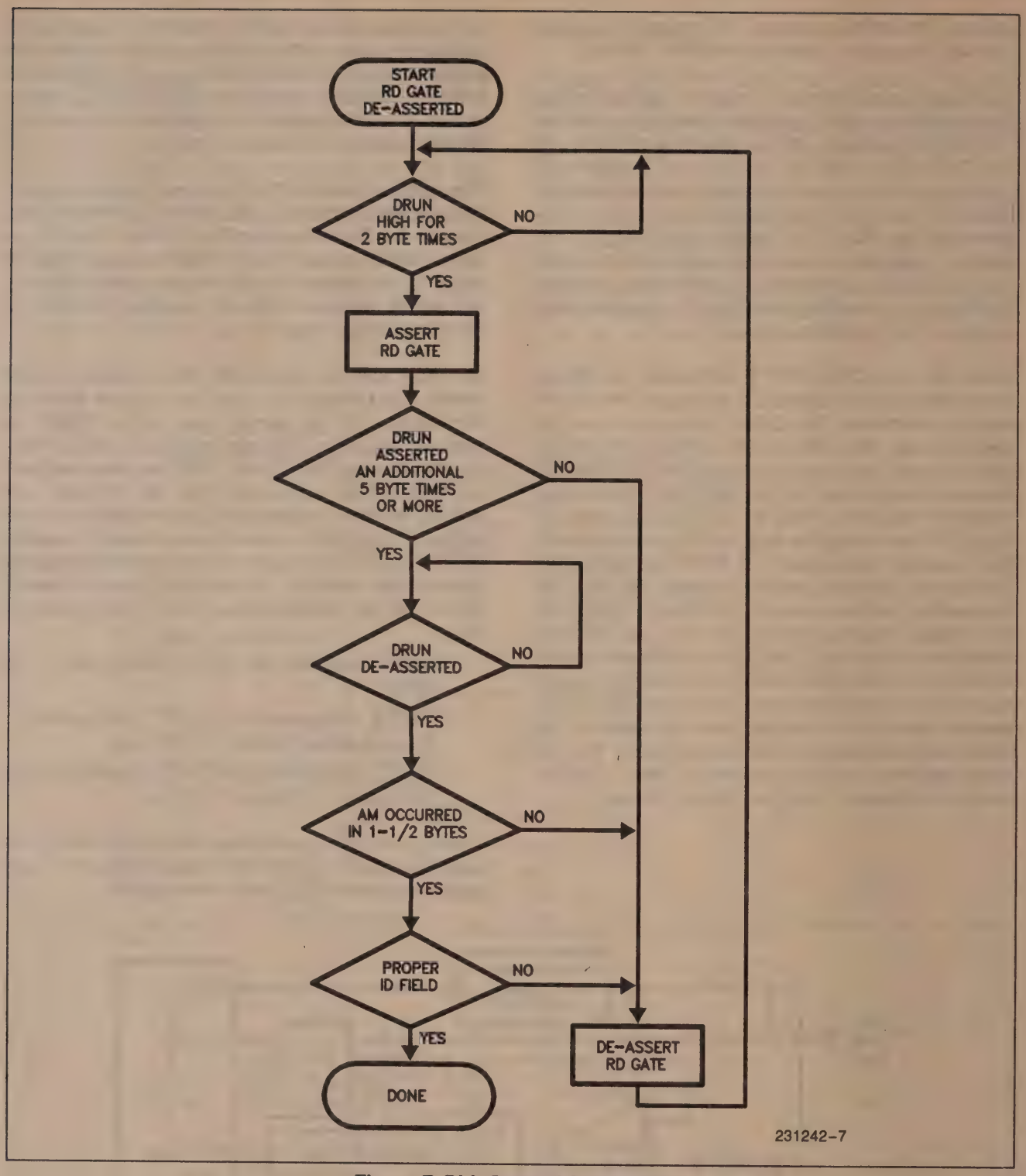


Figure 7. PLL Control Sequence

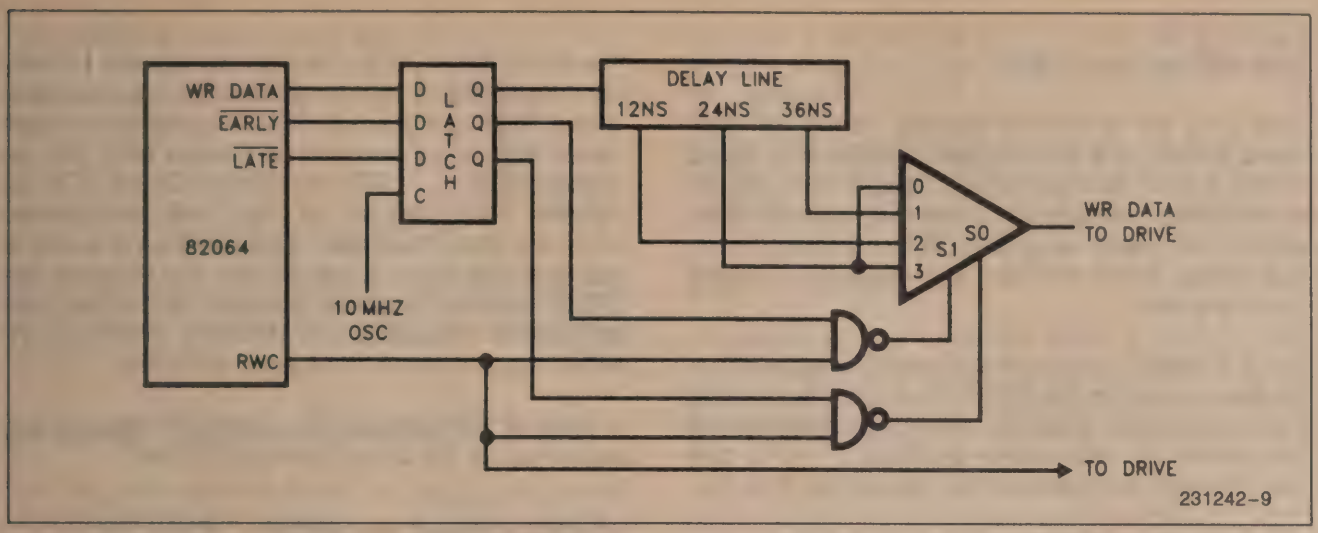


Figure 8. Write Precompensation Circuit

TASK REGISTER FILE

The Task Register File is a bank of nine registers used to hold parameter information pertaining to each command, status information, and the command itself. These registers and their addresses are:

A2	A1	A0	READ	WRITE
0	0	0	BUS TRI-STATED	BUS TRI-STATED
0	0	1	ERROR REGISTER	REDUCE WRITE CURRENT
0	1	0	SECTOR COUNT	SECTOR COUNT
0	1	1	SECTOR NUMBER	SECTOR NUMBER
1	0	0	CYLINDER LOW	CYLINDER LOW
1	0	1	CYLINDER HIGH	CYLINDER HIGH
1	1	0	SDH	SDH
1	1	1	STATUS	COMMAND

NOTE:
These registers are not cleared by $\overline{\text{RESET}}$ being asserted.

ERROR REGISTER

This read only register contains specific error information after the termination of a command. The bits are defined as follows:

7	6	5	4	3	2	1	0
BB	CRC/ECC	0	ID	0	AC	TK000	DAM

Bit 7 - Bad Block Detect (BB)

This bit is set when an ID field has been encountered that contains a bad block mark. It is used for bad sector mapping.

Bit 6 - CRC/ECC Data Field Error (CRC/ECC)

When in the CRC mode (SDH register, bit 7 = 0), this bit is set when a CRC error occurs in the data field. When retries are enabled, ten more attempts are made to read the sector correctly. If none of these attempts are successful bit 0 in the STATUS register is also set. If one of the attempts is successful, the CRC/ECC error bit remains set to inform the host that a marginal condition exists; however, bit 0 in the STATUS register is not set.

When in the ECC mode (SDH register, bit 7 = 1), this bit is set when the first non-zero syndrome is detected. When retries are enabled, up to ten attempts are made to correct the error. If the error is successfully corrected, this bit remains set; however, bit 2 of the STATUS register is also set to inform the host that the error has been corrected. If the error is not correctable, the CRC/ECC error bit remains set and bit 0 of the STATUS register is also set.

The data may be read even if uncorrectable errors exist.

NOTE: If the long mode (L) bit is set in the READ or WRITE command, no error checking is performed.

Bit 5 - Reserved

Not used. Forced to zero.

Bit 4 - ID Not Found (ID)

This bit is set to indicate that the correct cylinder, head, sector, or size parameter could not be found, or that a CRC error occurred in the ID field. This bit is set on the first failure and remains set even if the error is recovered on a retry. When recovery is unsuccessful, the Error bit (bit 0) of the STATUS register is also set.

For a SCAN ID command with retries enabled (T = 0), the Error bit in the STATUS register is set after ten unsuccessful attempts have been made to find the correct ID. With retries disabled (T = 1), only two attempts are made before setting the Error bit.

For a READ or WRITE command with retries enabled (T = 0), ten attempts are made to find the correct ID field. If there is still an error on the tenth try, an auto-scan and auto-seek are performed. Then ten more retries are made before setting the Error bit. When retries are disabled (T = 1), only two tries are made. No auto-scan or auto-seek operations are performed.

Bit 3 - Reserved

Not used. Forced to zero.

Bit 2 - Aborted Command (AC)

Command execution is aborted and this bit is set if a command was issued while DRDY is deasserted or WR FAULT is asserted. This bit will also be set if an undefined command is written to the COMMAND register; however, an implied seek will be executed.

Bit 1 - Track 000 Error (TK000)

This bit is set during the execution of a RESTORE command if the TRACK 000 pin has not gone active after the issuance of 2047 step pulses.

Bit 0 - Data Address Mark (DAM) Not Found

This bit is set during the execution of a READ SECTOR command if the DAM is not found following the proper sector ID.

REDUCE WRITE CURRENT REGISTER

This register is used to define the cylinder number where the RWC output (Pin 33) is asserted.

7	6	5	4	3	2	1	0
CYLINDER NUMBER ÷ 4							

The value (00-FFH) loaded into this cylinder is internally multiplied by four to specify the actual cylinder where RWC is asserted. Thus a value of 01H will cause RWC to be asserted on cylinder 04H, 02H on cylinder 08H, . . . , 9CH on cylinder 270H, 9DH on cylinder 274H, and so on. RWC will be asserted when the present cylinder is greater than or equal to four times the value of this register. For example, the ST506 interface requires precomp on cylinder 80H and above. Therefore, the REDUCE WRITE CURRENT register should be loaded with 20H.

A value of FFH causes RWC to remain deasserted, regardless of the actual cylinder number.

SECTOR COUNT REGISTER

This register is used to define the number of sectors that need to be transferred to the buffer during a READ MULTIPLE SECTOR or WRITE MULTIPLE SECTOR command.

7	6	5	4	3	2	1	0
NUMBER OF SECTORS							

The value contained in the register is decremented after each sector is transferred to/from the sector buffer. A zero represents a 256 sector transfer, a one a one sector transfer, etc. This register is a "don't care" when single sector commands are specified.

SECTOR NUMBER REGISTER

This register holds the sector number of the desired sector.

7	6	5	4	3	2	1	0
SECTOR NUMBER							

For a multiple sector command, it specifies the first sector to be transferred. It is incremented after each sector is transferred to/from the sector buffer. The SECTOR NUMBER register may contain any value from 0 to 255.

The SECTOR NUMBER register is also used to program the Gap 1 and Gap 3 lengths to be used when formating a disk. See the WRITE FORMAT command description for further explanation.

CYLINDER NUMBER LOW REGISTER

This register holds the lower byte of the desired cylinder number.

7	6	5	4	3	2	1	0
LS BYTE OF CYL. NUMBER							

It is used with the CYLINDER NUMBER HIGH register to specify the desired cylinder number over a range of 0 to 2047.

CYLINDER NUMBER HIGH REGISTER

This register holds the three most significant bits of the desired cylinder number.

7	6	5	4	3	2	1	0
x	x	x	x	x	#	#	#

The CYLINDER NUMBER LOW/HIGH register pair determine where the R/W heads are to be positioned. The host writes the desired cylinder number into these registers. Internal to the 82064 is another pair of registers that hold the present head location. When any command other than a RESTORE is executed, the internal head location registers are compared to the CYLINDER NUMBER registers to determine how many cylinders to move the heads and in what direction.

The internal head location registers are updated to equal the CYLINDER NUMBER registers after the completion of the seek.

When a RESTORE command is executed, the internal head location registers are reset to zero while DIR and STEP move the heads to track zero.

SECTOR/DRIVE/HEAD (SDH) REGISTER

The SDH register contains the desired sector size, drive number, and head parameters. The format is shown in Figure 9. The EXT bit (bit 7) is used to select between the CRC or ECC mode. When bit 7 = 0 the ECC mode is selected for the data field. When bit 7 = 1 the CRC mode is selected.

The SDH byte written in the ID field of the disk by the FORMAT command is different than the SDH register contents. The recorded SDH byte does not have the drive number recorded, but does have the bad block mark written. The format of the SDH byte written on the disk is:

7	6	5	4	3	2	1	0
BAD B.	SIZE		0	0	HEAD		

STATUS REGISTER

The status register is used to inform the host of certain events performed by the 82064, as well as reporting status from the drive control lines. Reading the STATUS register deasserts INTRQ. The format is:

7	6	5	4	3	2	1	0
BUSY	READY	WF	SC	DRQ	DWC	CIP	ERR

Bit 7 - Busy

This bit is asserted when a command is written into the COMMAND register and, except for the READ command, is deasserted at the end of the command. When executing a READ command, Busy will be deasserted when the sector buffer is full. Commands should not be loaded into the COMMAND register when Busy is set. When the Busy bit is set, no other bits in the STATUS or ERROR registers are valid.

Bit 6 - Ready

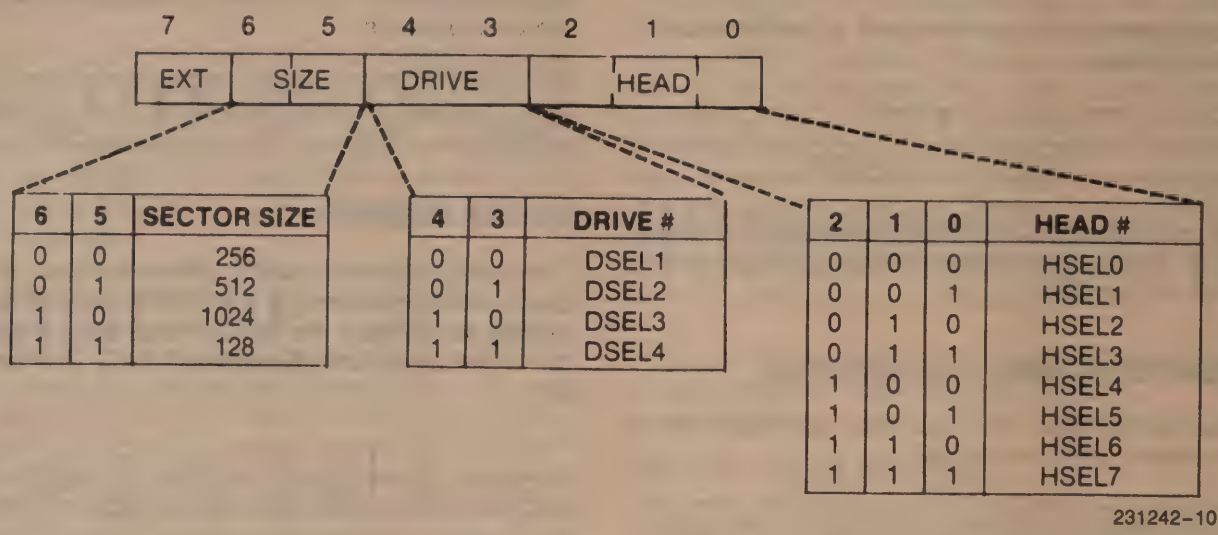
This bit reflects the status of DRDY (pin 28). When this bit equals zero, the command is aborted and the status of this bit is latched.

Bit 5 - Write Fault (WF)

This bit reflects the status of WR FAULT (pin 30). When this bit equals one the command is aborted, INTRQ is asserted, and the status of this bit is latched.

Bit 4 - Seek Complete (SC)

This bit reflects the status of SC (pin 32). When a seek or implied seek has been initiated by a command, execution of the command pauses until the seek is complete. This bit is latched after an aborted command error.



6	5	SECTOR SIZE
0	0	256
0	1	512
1	0	1024
1	1	128

4	3	DRIVE #
0	0	DSEL1
0	1	DSEL2
1	0	DSEL3
1	1	DSEL4

2	1	0	HEAD #
0	0	0	HSEL0
0	0	1	HSEL1
0	1	0	HSEL2
0	1	1	HSEL3
1	0	0	HSEL4
1	0	1	HSEL5
1	1	0	HSEL6
1	1	1	HSEL7

NOTE:
Drive select and head select lines must be generated externally. Figure 3 represents one method of achieving this.

Figure 9. SDH Register Format

Bit 3 - Data Request (DRQ)

The DRQ bit reflects the status of BDRQ (pin 36). It is asserted when the sector buffer must be written into or read from. DRQ and BDRQ remain asserted until BRDY indicates that the sector buffer has been filled or emptied, depending upon the command. BDRQ can be used for DMA interfacing, while DRQ is used in a programmed I/O environment.

Bit 2 - Data Was Corrected (DWC)

When set, this bit indicates that an ECC error has been detected during a read operation, and that the data in the sector buffer has been corrected. This provides the user with an indication that there may be a marginal condition within the drive before the errors become uncorrectable. This bit is forced to zero when not in the ECC mode.

Bit 1 - Command In Progress (CIP)

When this bit is set a command is being executed and a new command should not be loaded. Although a command is being executed, the sector buffer is still available for access by the host. When the 82064 is no longer Busy (bit 7 = 0) the STATUS register can be read. If other registers are read while CIP is set the contents of the STATUS register will be returned.

Bit 0 - Error

This bit is set whenever any bits in the ERROR register are set. It is the logical 'or' of the bits in the ERROR register and may be used by the host processor to quickly check for nonrecoverable errors. The host must read the ERROR register to determine what type of error occurred. This bit is reset when a new command is written into the COMMAND register.

COMMAND REGISTER

The command to be executed is written into this write-only register:

7	6	5	4	3	2	1	0
COMMAND							

The command sets Busy and CIP, and begins to execute as soon as it is written into this register. Therefore, all necessary information should be loaded into the Task Register File prior to entering the command. Any attempt to write a register will be ignored until command execution has terminated, as indicated by the CIP bit being cleared. INTRQ is deasserted when the COMMAND register is written.

COMMAND	7	6	5	4	3	2	1	0
RESTORE	0	0	0	1	R3	R2	R1	R0
SEEK	0	1	1	1	R3	R2	R1	R0
READ SECTOR	0	0	1	0	I	M	L	T
WRITE SECTOR	0	0	1	1	0	M	L	T
SCAN ID	0	1	0	0	0	0	0	T
WRITE FORMAT	0	1	0	1	0	0	0	0
COMPUTE CORRECTION	0	0	0	0	1	0	0	0
SET PARAMETER	0	0	0	0	0	0	0	S

R_{3-0} = Stepping Rate Field

For 5 MHz WR CLOCK:

R_{3-0} = 0000 35 μs
0001 0.5 ms
0010 1.0 ms
0011 1.5 ms
0100 2.0 ms
0101 2.5 ms
0110 3.0 ms
0111 3.5 ms
1000 4.0 ms
1001 4.5 ms
1010 5.0 ms
1011 5.5 ms
1100 6.0 ms
1101 6.5 ms
1110 3.2 μs
1111 16 μs

I = Interrupt Control

I = 0 INTRQ occurs with BDRQ/DRQ indicating the sector buffer is full. Valid only when M = 0.

I = 1 INTRQ occurs when the command is completed and the host has read the sector buffer.

M = Multiple Sector Flag

M = 0 Transfer one sector. Ignore the SECTOR COUNT register.

M = 1 Transfer multiple sectors.

L = Long Mode

L = 0 Normal mode. Normal CRC or ECC functions are performed.

L = 1 Long mode. No CRC or ECC bytes are developed or error checking performed on the data field. The 82064 appends the four additional bytes supplied by the host or disk to the data field.

T = Retry Enable

T = 0 Enable retries.

T = 1 Disable retries.

S = Error Correction Span

S = 0 5-bit span.

S = 1 11-bit span.

RESTORE COMMAND

The RESTORE command is used to position the R/W heads over track zero. It is usually issued by the host when a drive has just been turned on. The 82064 forces an auto-restore when a FORMAT command has been issued following a drive number change.

The actual step rate used for the RESTORE command is determined by the seek complete time. A step pulse is issued and the 82064 waits for a rising edge on the SC line before issuing the next pulse. If the rising edge of SC has not occurred within ten revolutions (INDEX pulses) the 82064 switches to sensing the level of SC. If after 2047 step pulses the TRACK 000 line does not go active the 82064 will set the TRACK 000 bit in the ERROR register, assert INTRQ, and terminate execution of the command. An interrupt will also occur if WR FAULT is asserted on DRDY is deasserted at any time during execution.

The rate field specified (R_{3-0}) is stored in an internal register for future use in commands with implied seeks.

A flowchart of the RESTORE command is shown in Figure 10.

SEEK COMMAND

The SEEK command can be used for overlapping seeks on multiple drives. The step rate used is taken from the Rate Field of the command, and is stored in an internal register for future use by those commands with implied seek capability.

The direction and number of step pulses needed are calculated by comparing the contents of the CYLINDER NUMBER registers in the Task Register File to the present cylinder position stored internally. After all the step pulses have been issued the present cylinder position is updated, INTRQ is asserted, and the command terminated.

If DRDY is deasserted or WR FAULT is asserted during the execution of the command, INTRQ is asserted and the command aborts setting the AC bit in the ERROR register.

If an implied seek is performed, the step rate indicated by the rate field is used for all but the last step pulse. On the last pulse, the command execution continues until the rising edge of SC is detected. If 10 INDEX pulses are received without a rising edge of SC, the 82064 will switch to sensing the level of SC.

A flowchart of the SEEK command flow is shown in Figure 11.

READ SECTOR

The READ SECTOR command is used to transfer one or more sectors of data from the disk to the sector buffer. Upon receipt of the command, the 82064 checks the CYLINDER NUMBER LOW/HIGH register pair against the internal cylinder position register to see if they are equal. If not, the direction and number of steps calculation takes place, and a seek is initiated. As stated in the description of the SEEK command, if an implied seek occurs, the step rate specified by the rate field is used for all but the last step pulse. On the last step pulse the seek continues until the rising edge of SC is detected.

If the 82064 detects a change in the drive number since the last command, an auto-scan ID is performed. This updates the internal cylinder position register to reflect the current drive before the seek begins.

After the 82064 senses SC (with or without an implied seek) it must find an ID field with the correct cylinder number, head, sector size, and CRC. If retries are enabled (T = 0), ten attempts are made to find the correct ID field. If there is still an error on the tenth try, an auto-scan ID and auto-seek are performed. Then ten more retries are attempted before setting the ID Not Found error bit. When retries are disabled (T = 1) only two tries are made. No auto-scan or auto-seek operations are performed.

When the data address mark (DAM) is found, the 82064 is ready to transfer data into the sector buffer. When the disk has filled the sector buffer, the 82064 asserts BDRQ and DRQ and then checks the I flag. If I = 0, INTRQ is asserted, signaling the host to read the contents of the sector buffer. If I = 1, INTRQ occurs after the host has read the sector buffer and the command has terminated. If after successfully reading the ID field, the DAM is not found the DAM Not Found bit in the ERROR register is set.

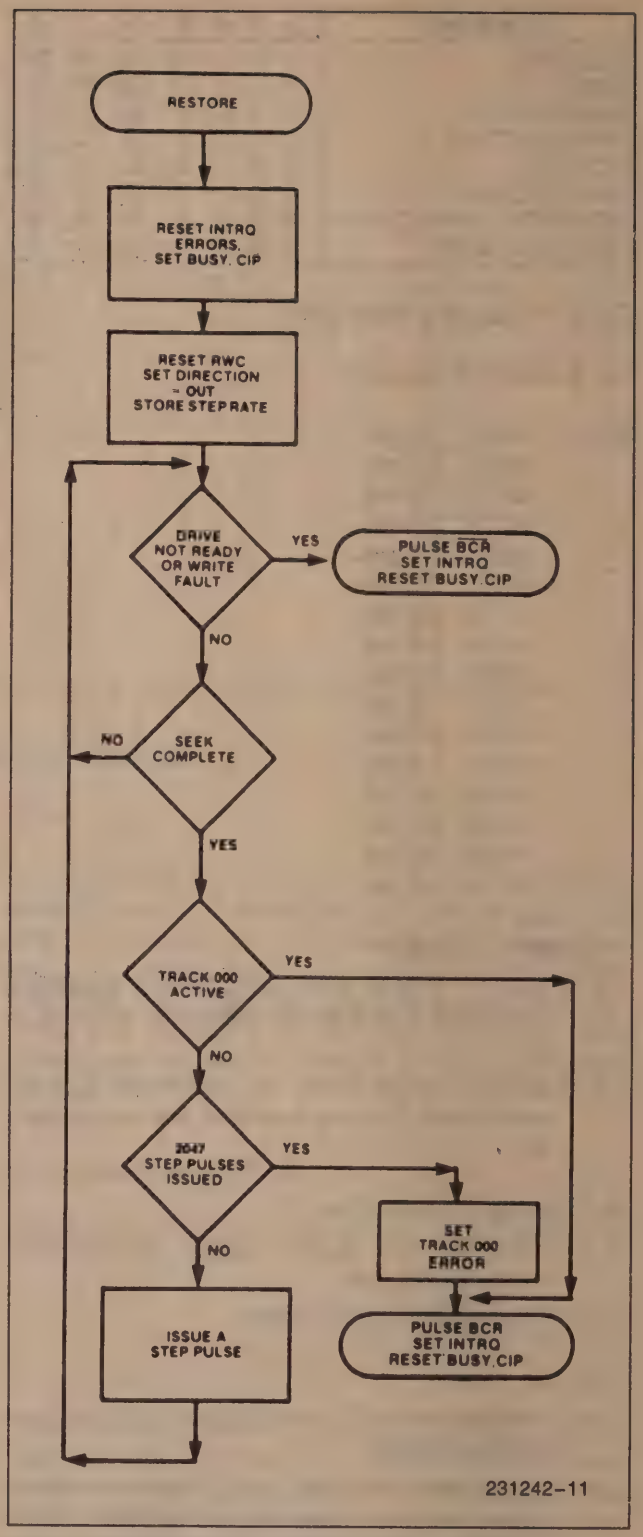


Figure 10. Restore Command Flow

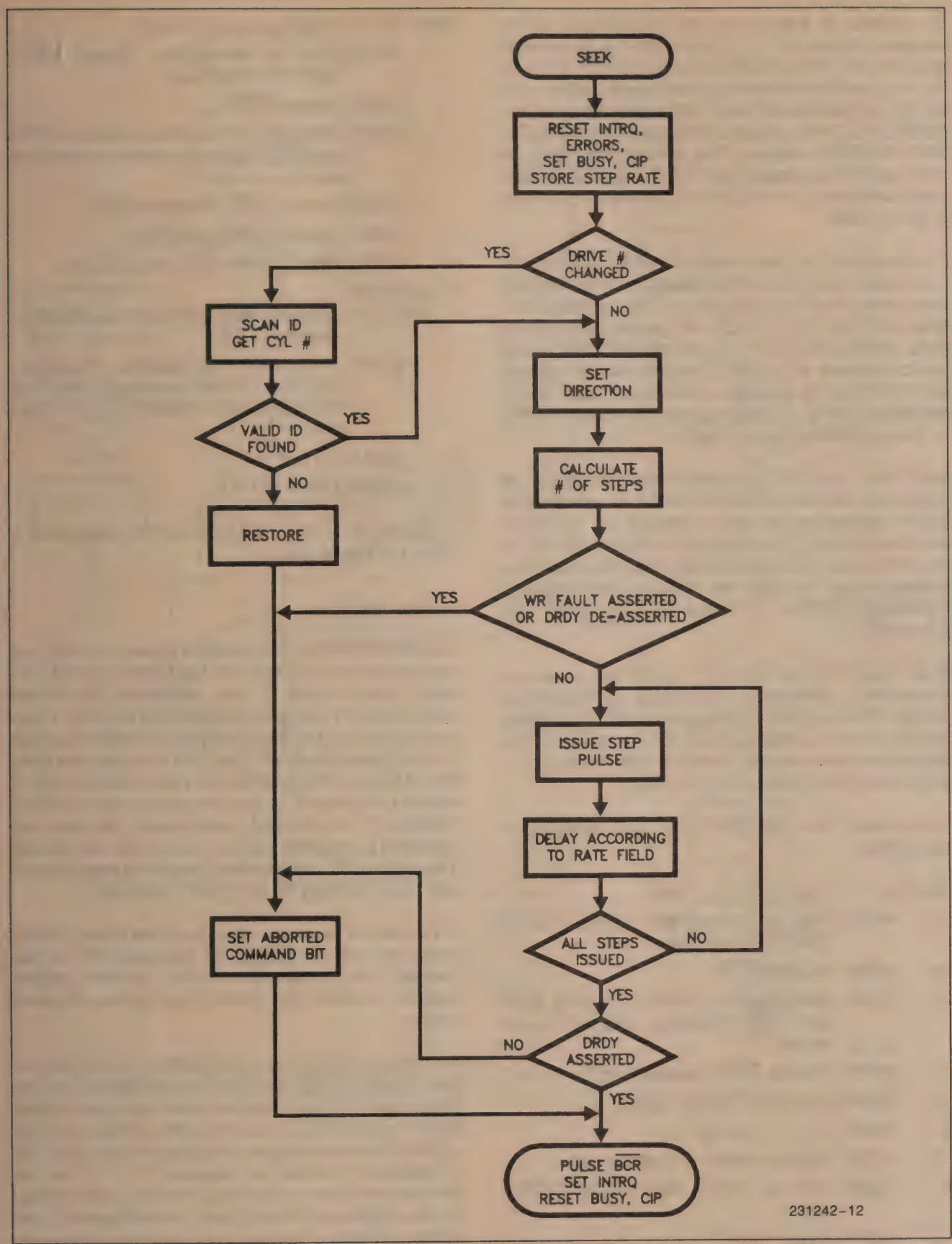


Figure 11. Seek Command Flow

An optional M flag can be set for multiple sector transfers. When M = 0, one sector is transferred and the SECTOR COUNT register is ignored. When M = 1, multiple sectors are transferred. After each sector is transferred, the 82064 decrements the SECTOR COUNT register and increments the SECTOR NUMBER register. The next logical sector is transferred regardless of any interleave. Sectors are numbered during the FORMAT command by a byte in the ID field.

For the 82064 to make multiple sector transfers to the sector buffer, the BRDY signal must be toggled from low to high for each sector. The transfers continue until the SECTOR COUNT register equal zero. If the SECTOR COUNT is not zero (indicating more sectors remain to be read), and the sector buffer is full, BDRQ will be asserted and the host must unload the sector buffer. Once this occurs, the sector buffer is free to accept the next sector.

WR FAULT and DRDY are monitored throughout the command execution. If WR FAULT is asserted or DRDY is deasserted, the command will terminate and the Aborted Command bit in the ERROR register will be set. For a description of the error checking procedure on the data field see the explanation in the section entitled "CRC and ECC Generator and Checker."

Both the READ and WRITE commands feature a "simulated completion" to ease programming. BDRQ, DRQ, and INTRQ are generated in a normal manner upon detection of an error condition. This allows the same program flow for successful or unsuccessful completion of a command.

In summary then, the READ SECTOR operation is as follows:

When M = 0 (Single Sector Read)

1. HOST: Sets up parameters. Issues READ SECTOR command.
2. 82064: Asserts $\overline{BCR}$.
3. 82064: Finds sector specified. Asserts $\overline{BCR}$ and $\overline{BCS}$. Transfers data to sector buffer.
4. 82064: Asserts $\overline{BCR}$. Deasserts $\overline{BCS}$.
5. 82064: Asserts BDRQ and DRQ.
6. 82064: If I = 1 then go to 9.
7. HOST: Read contents of sector buffer.
8. 82064: Wait for BRDY, then assert INTRQ. End.
9. 82064: Assert INTRQ.
10. HOST: Read contents of sector buffer. End.

When M = 1 (Multiple Sector Read)

1. HOST: Sets up parameters. Issues READ SECTOR command.
2. 82064: Asserts $\overline{BCR}$.
3. 82064: Finds sector specified. Asserts $\overline{BCR}$ and $\overline{BCS}$. Transfers data to sector buffer.
4. 82064: Asserts $\overline{BCR}$. Deasserts $\overline{BCS}$.
5. 82064: Asserts BDRQ and DRQ.
6. HOST: Reads contents of sector buffer.
7. SECTOR BUFFER: Indicates data has been transferred by asserting BRDY.
8. 82064: When BRDY is asserted, decrement SECTOR COUNT, increment SECTOR NUMBER. If SECTOR COUNT = 0, go to 11.
9. 82064: Go to 2.
10. 82064: Assert INTRQ.

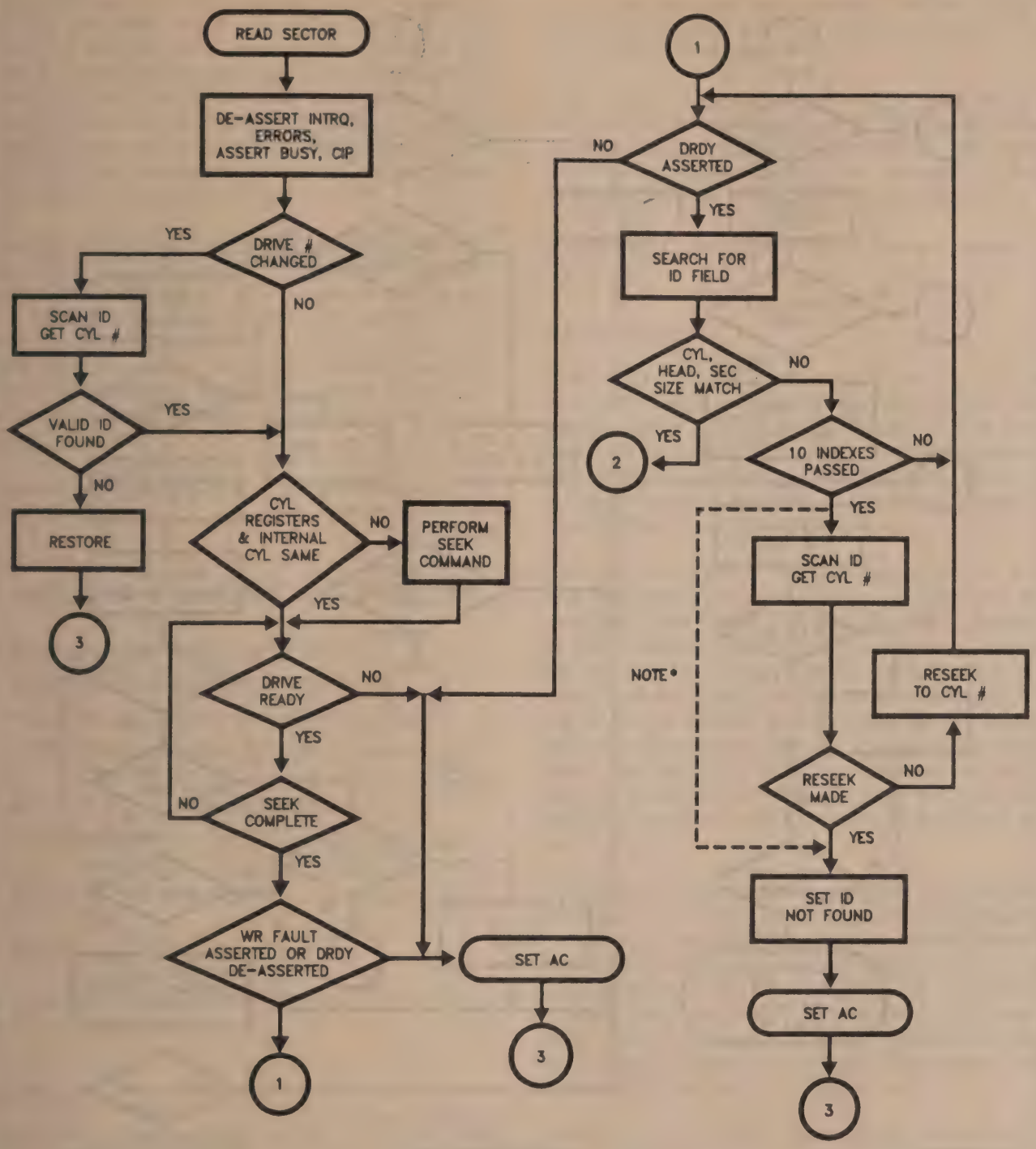
A flowchart of the READ SECTOR command is shown in Figure 12.

WRITE SECTOR

The WRITE SECTOR command is used to write one or more sectors of data from the sector buffer to the disk. Upon receipt of the command, the 82064 checks the CYLINDER NUMBER LOW/HIGH register pair against the internal cylinder position register to see if they are equal. If not, the direction and number of steps calculation takes place, and a seek is initiated. As stated in the description of the SEEK command, if an implied seek occurs, the step rate specified by the rate field is used for all but the last step pulse. On the last step pulse the seek continues until the rising edge of SC is detected.

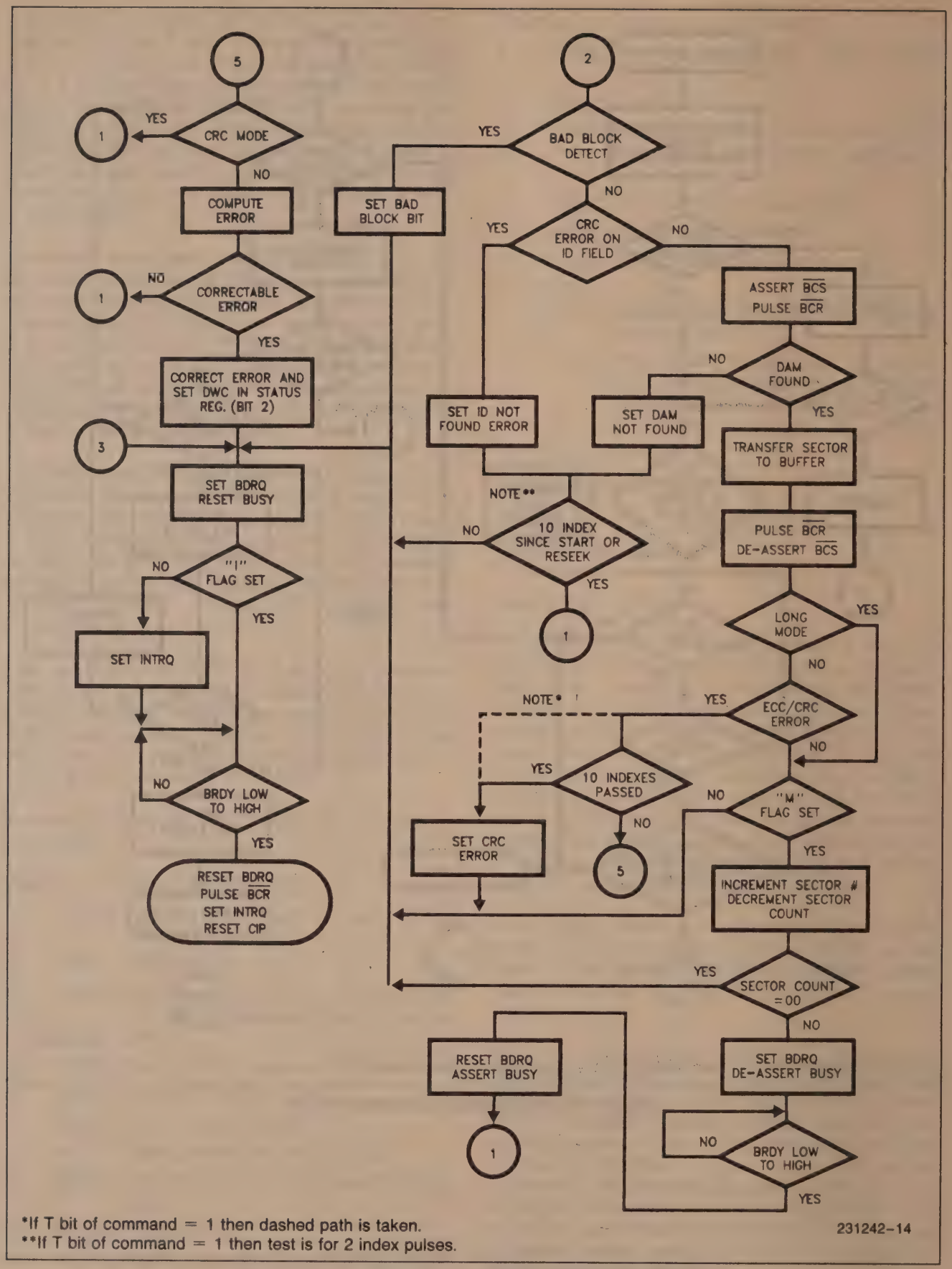
If the 82064 detects a change in the drive number since the last command, an auto-scan ID is performed. This updates the internal cylinder position register to reflect the current drive before the seek begins.

After the 82064 senses SC (with or without an implied seek) BDRQ and DRQ are asserted and the host begins filling the sector buffer with data. When BRDY is asserted, a search for the ID field with the correct cylinder number, head, sector size, and CRC is initiated. If retries are enabled (T = 0), ten attempts are made to find the correct ID field. If there is still an error on the tenth try, an auto-scan ID and auto-seek are performed. Then ten more retries are attempted before setting the ID Not Found error bit. When retries are disabled (T = 1) only two tries are made. No auto-scan or auto-seek operations are performed.



*If T bit of command = 1 then dashed path is taken after 2 index pulses.

Figure 12a. Read Sector Command Flow



*If T bit of command = 1 then dashed path is taken.
**If T bit of command = 1 then test is for 2 index pulses.

231242-14

Figure 12b. Read Sector Command Flow (Continued)

When the correct ID is found, WR GATE is asserted and data is written to the disk. When the CRC/ECC bit (SDH Register, bit 7) is zero, the 82064 generates a two byte CRC character to be appended to the data. When the CRC/ECC bit is one, four ECC bytes replace the CRC character. When L = 1, the polynomial generator is inhibited and neither CRC or ECC bytes are generated. Instead four bytes of data supplied by the host are written.

During a WRITE MULTIPLE SECTOR command (M = 1), the SECTOR NUMBER register is incremented and the SECTOR COUNT register is decremented. If BRDY is asserted after the first sector is read from the sector buffer, the 82064 continues to read data from the sector buffer for the next sector. If BRDY is deasserted, the 82064 asserts BDRQ and waits for the host to place more data in the sector buffer.

In summary then, the WRITE SECTOR operation is as follows:

When M = 0,1

1. HOST: Sets up parameters. Issues READ SECTOR command.
2. 82064: Asserts BDRQ and DRQ.
3. HOST: Loads sector buffer with data.
4. 82064: Waits for rising edge of BRDY.
5. 82064: Finds specified ID field. Writes sector to disk.
6. 82064: If M = 0, asserts INTRQ. End.
7. 82064: Increments SECTOR NUMBER. Decrements SECTOR COUNT.
8. 82064: IF SECTOR COUNT = 0, assert INTRQ. End.
9. 82064: Go to 2.

A flowchart of the WRITE SECTOR command is shown in Figure 13.

SCAN ID

The SCAN ID command is used to update the SDH, SECTOR NUMBER, and CYLINDER NUMBER LOW/HIGH registers.

After the command is loaded, the SC line is sampled until it is valid. The DRDY and WR FAULT lines are also monitored throughout execution of the command. If a fault occurs the command is aborted and the appropriate error bits are set. When the first ID field is found, the ID information is loaded into the SDH, SECTOR NUMBER, and CYLINDER NUMBER registers. The internal cylinder position register is also updated. If this is an auto-scan caused by a change in drive numbers, only the internal position register is updated. If a bad block is detected, the BAD BLOCK bit will also be set.

If an ID field is not found, or if a CRC error occurs, and if retries are enabled (T = 0), ten attempts are made to read it. If retries are disabled (T = 1), only two tries are made. There is no auto-seek in this command and the sector buffer is not disturbed.

A flowchart of the SCAN ID command is shown in Figure 14.

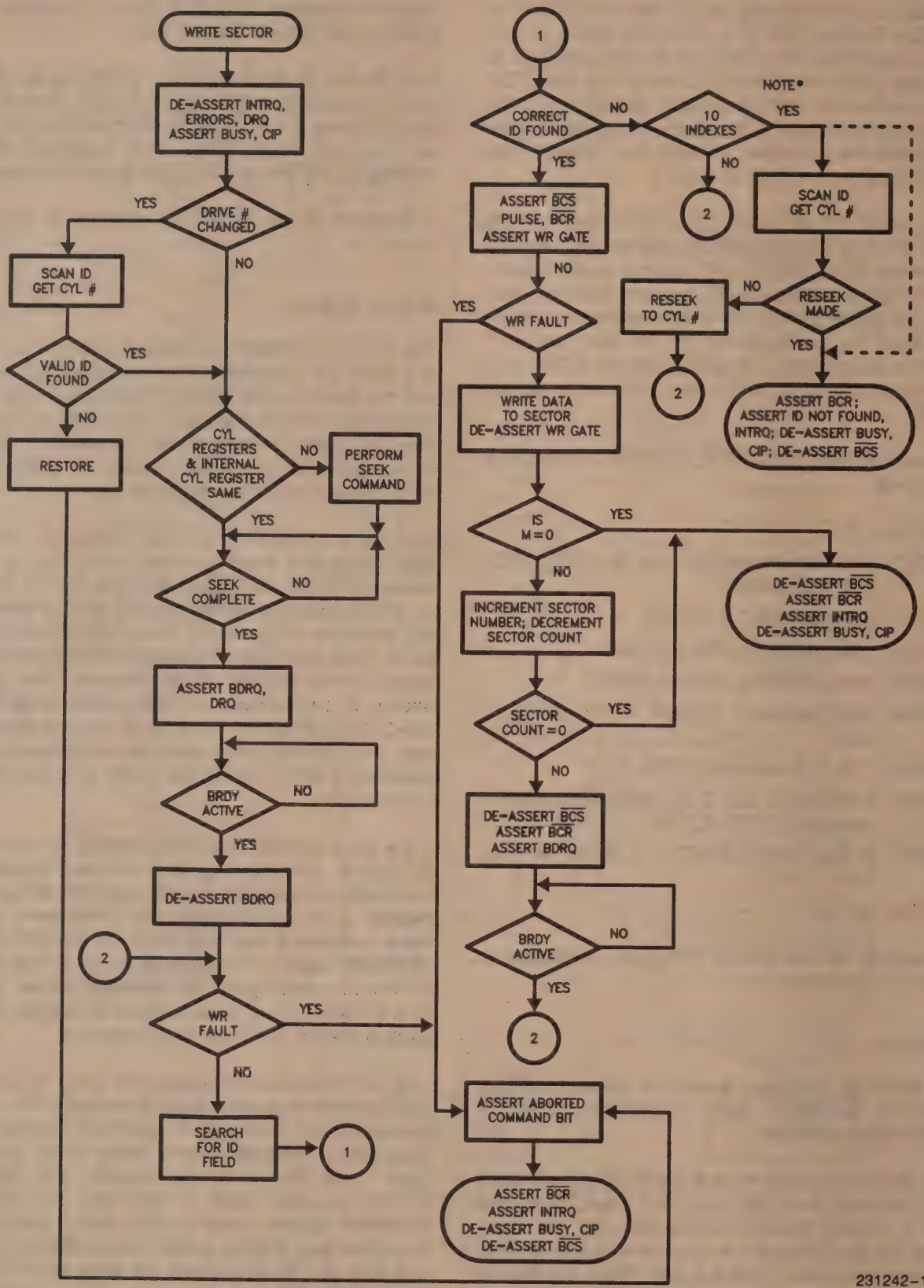
WRITE FORMAT

The WRITE FORMAT command is used to format one track using information in the Task Register File and the sector buffer. During execution of this command, the sector buffer is used for additional parameter information instead of data. Shown in Figure 15 is the contents of a sector buffer for a 32 sector track with an interleave factor of two.

Each sector requires a two byte sequence. The first byte designates whether a bad block mark is to be recorded in the sector's ID field. An 00H is normal; an 80H indicates a bad block mark for that sector. In the example of Figure 15, sector 04 will get a bad block mark recorded. The second byte indicates the logical sector number to be recorded. This allows sectors to be recorded with any interleave factor desired. The remaining memory in the sector buffer may be filled with any value; its only purpose is to generate a BRDY to tell the 82064 to begin formatting the track.

If the drive number has been changed since the last command, an auto-restore is initiated, positioning the heads to track 000. The internal cylinder position register is set to zero and the heads seek to the track specified in the Task Register File CYLINDER NUMBER register. This prevents an ID Not Found error from occuring due to an incompatible format, or the track having been erased. A normal implied seek is also in effect for this command.

The SECTOR COUNT register is used to hold the total number of sectors to be formatted (FFH = 255 sectors), while the SECTOR NUMBER register holds the number of bytes, minus three, to be used for Gap 1 and Gap 3. If, for example, the SECTOR COUNT register value is 02H and the SECTOR NUMBER register value is 00H, then 2 sectors are formatted and 3 bytes of 4EH are written for Gap 1 and Gap 3. The data fields are filled with FFH and the CRC or ECC is automatically generated and appended. After the last sector is written the track is filled with 4EH.



*If retries disabled then dashed path is taken after 2 index pulses.

Figure 13. Write Sector Command Flow

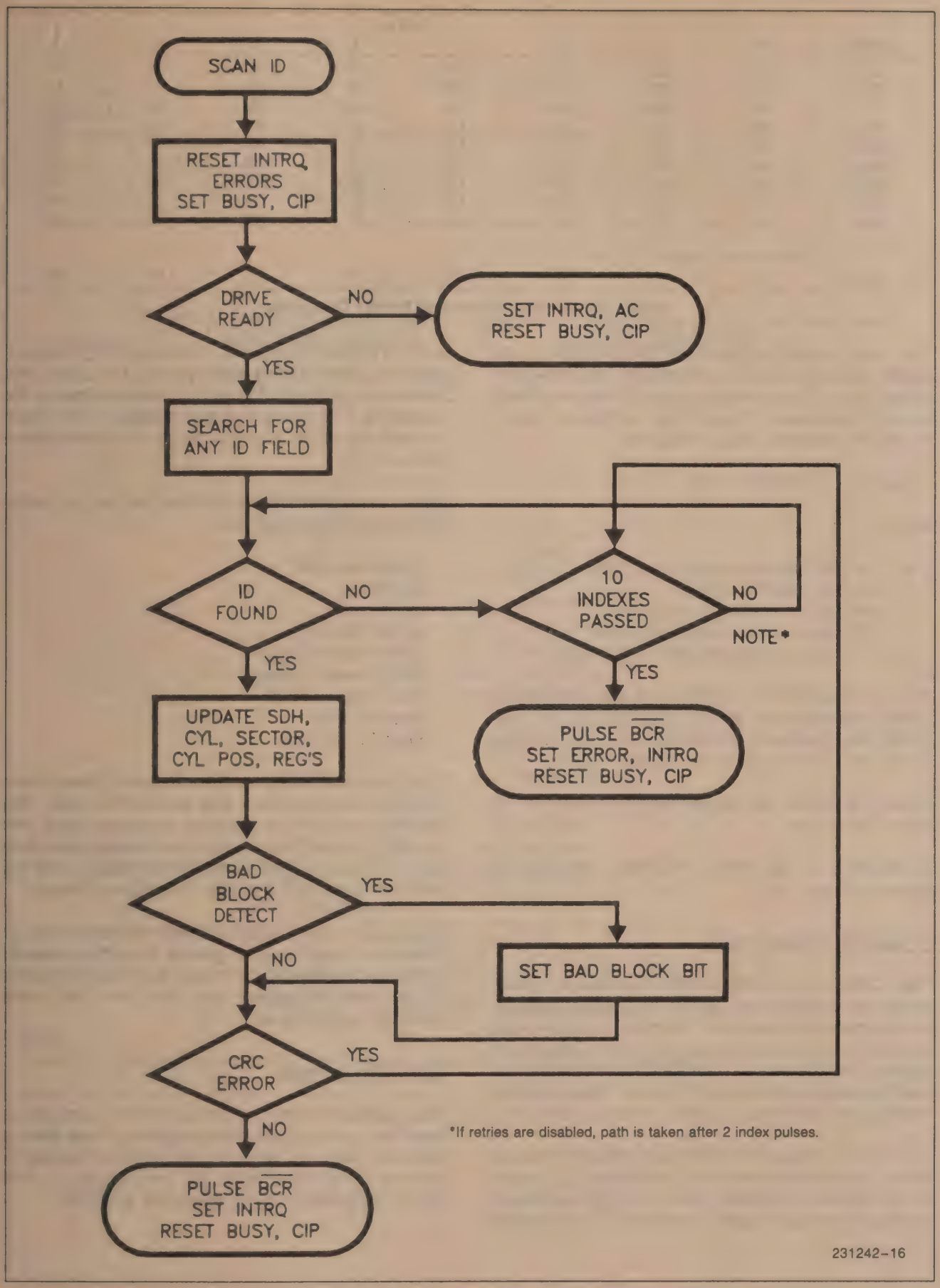


Figure 14. Scan ID Command Flow

ADDR	DATA 0	1	2	3	4	5	6	7
00	00	00	00	10	00	01	00	11
08	00	02	00	12	00	03	00	13
10	80	04	00	14	00	05	00	15
18	00	06	00	16	00	07	00	17
20	00	08	00	18	00	09	00	19
28	00	0A	00	1A	00	0B	00	1B
30	00	0C	00	1C	00	0D	00	1D
38	00	0E	00	1E	00	0F	00	1F
40	FF	FF	FF	FF	FF	FF	FF	FF
				:				
				:				
F0	FF	FF	FF	FF	FF	FF	FF	FF

Figure 15. Format Command Buffer Contents

The Gap 3 value is determined by the drive motor speed variation, data sector length, and the interleave factor. The interleave factor is only important when 1:1 interleave is used. The formula for determining the minimum Gap 3 length is:

$$\text{Gap 3} = (2*M*S)+K+E$$

where:

M = motor speed variation (e.g., 0.03 for + 3%)
S = sector length in bytes
K = 18 for an interleave factor of 1
0 for any other interleave factor
E = 2 if ECC is enabled (SDH register, bit 7 = 1)

As for all commands, if WR FAULT is asserted or DRDY is deasserted during execution of the command, the command terminates and the Aborted Command bit in the ERROR register is set.

Figure 16 shows the format which the 82064 will write on the disk.

A flowchart of the WRITE FORMAT command is shown in Figure 17.

COMPUTE CORRECTION

The COMPUTE CORRECTION command determines the location and pattern of a single burst error, but does not correct it. The host, using the data provided by the 82064, must perform the actual correction. The COMPUTE CORRECTION command is used following a data field ECC error. The command initiating the read must specify no retries (T = 1).

The COMPUTE CORRECTION command first writes the four syndrome bytes from the internal ECC register to the sector buffer. Then the ECC register is clocked. With each clock, a counter is incremented and the pattern examined. If the pattern is correctable, the procedure is stopped and the count and pattern are written to the sector buffer, following the syndrome. The process is also stopped if the count exceeds the sector size before a correctable pattern is found.

When the command terminates the sector buffer contains the following data:

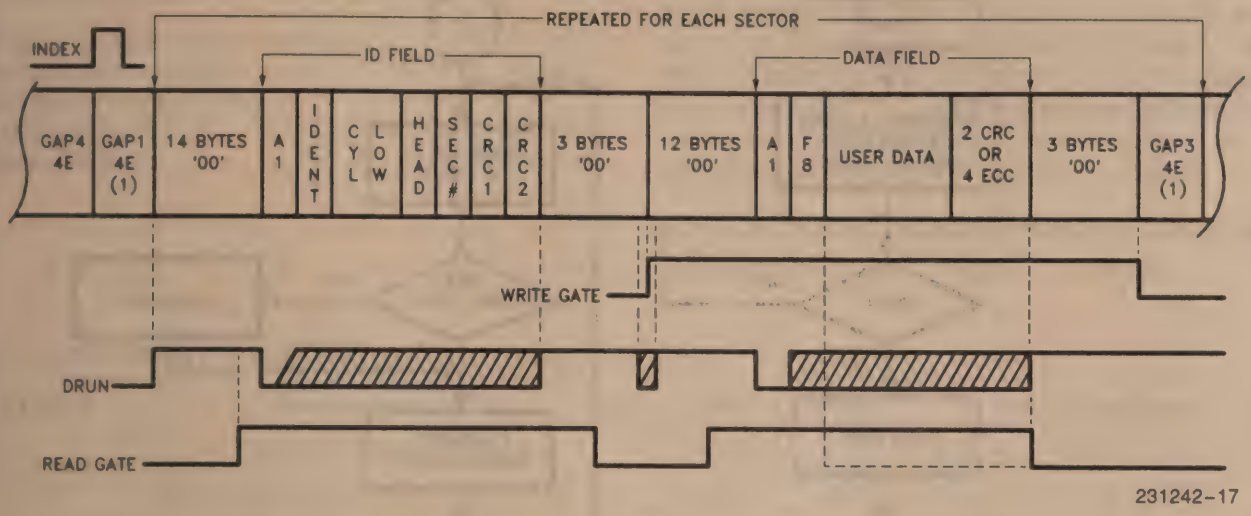
Syndrome MSB
Syndrome
Syndrome
Syndrome LSB
Error Pattern Offset
Error Pattern Offset
Error Pattern MSB
Error Pattern
Error Pattern LSB

As an example, when the Error Pattern Offset is zero the following procedure may correct the error. The first data byte of the sector is exclusive OR'd with the MSB of the Error Pattern, the second data byte with the second byte of the Error Pattern, and the third data byte with the LSB of the Error Pattern.

If the sector buffer count exceeds the sector size, or if the error burst length is greater than that selected by the Set Parameter command, the ECC/CRC error in the ERROR register and the Error bit in the STATUS register is set.

SET PARAMETER

This command selects the correction span to be used for the error correction process. A 5-bit span is selected when bit zero of the command equals 0, and an 11-bit span when bit zero equals 1. The 82064 defaults to a 5-bit span after a RESET.



ID FIELD		
A1	=	A1H with 0AH Clock
IDENT	=	Bits 3, 1, 0 = Cylinder High FE = 0-255 Cylinders FF = 256-511 Cylinders FC = 512-767 Cylinders FD = 768-1023 Cylinders F6 = 1024-1279 Cylinders F7 = 1280-1535 Cylinders F4 = 1536-1791 Cylinders F5 = 1792-2047 Cylinders
HEAD	=	Bits 0, 1, 2 = Head Number Bits 3, 4 = 0 Bits 5, 6 = Sector Size Bit 7 = Bad Block Mark
Sec #	=	Logical Sector Number
DATA FIELD		
A1	=	A1H with 0AH clock
F8	=	Data Address Mark; Normal Clock
USER	=	Data Field 128 to 1024 Bytes

NOTE:
1. GAP 1 and 3 length determined by Sector Number Register contents during formatting.

Figure 16. Track Format

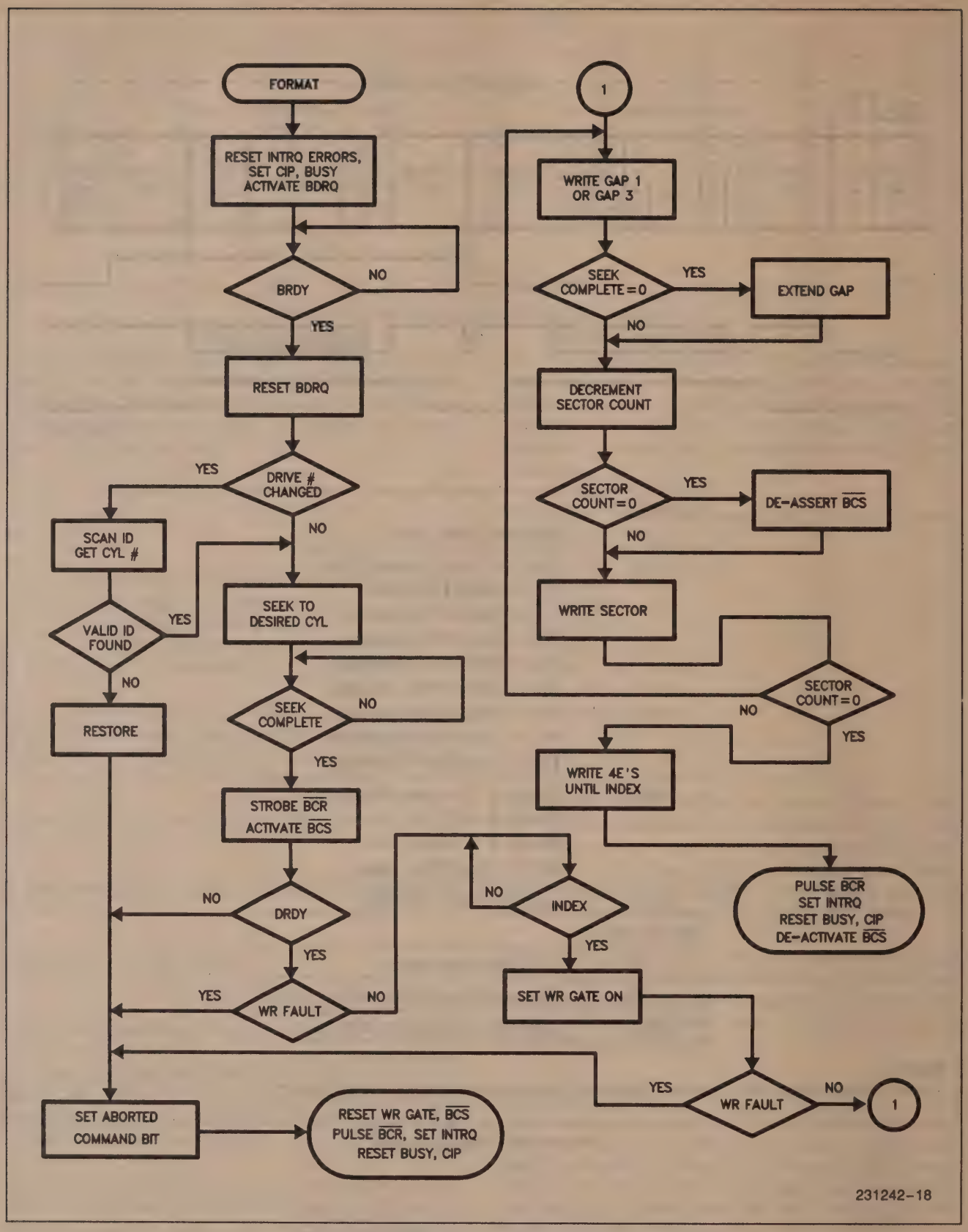


Figure 17. Write Format Command Flow

ELECTRICAL CHARACTERISTICS
ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias .....0°C to 70°C
Storage Temperature .....−65°C to +150°C
Voltage on any pin with
respect to GND.....−0.5V to +7V
Power Dissipation.....1.5 Watt

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

NOTICE: Specifications contained within the following tables are subject to change.

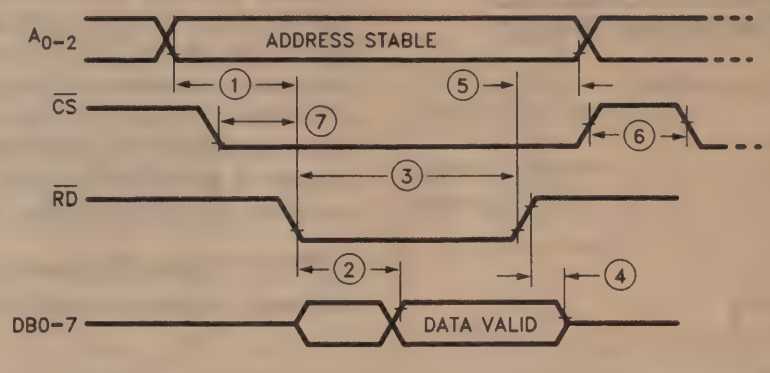
D.C. CHARACTERISTICS (T_A = 0°C to 70°C; V_{CC} = +5V ± 10%; GND = 0V)

Symbol	Parameter	Min	Max	Units	Test Conditions
I_{IL}	Input Leakage Current		±10	μA	V_{IN} = V_{CC} to 0V
I_{OFL}	Output Leakage Current		±10	μA	V_{OUT} = V_{CC} to 0.45V
V_{IH}	Input High Voltage	2.0		V	
V_{IL}	Input Low Voltage		0.8	V	
V_{OH}	Output High Voltage	2.4		V	I_{OH} = −100 μA
V_{OL}	Output Low Voltage		0.40 0.45	V	I_{OL} = 1.6 mA 6.0 mA P21, 22, 23
I_{CC}	Supply Current		160	mA	All Outputs Open
C_{IN}	Input Capacitance		10	pF	f_c = 1 MHz
$C_{I/O}$	I/O Capacitance		20	pF	Unmeasured pins returned to GND
	For Pins 25, 34, 37, 39 (WR CLOCK, DRUN, READ DATA, READ CLOCK)				
TRS	Rise Time		30	ns	0.9V to 4.2V

A.C. CHARACTERISTICS (T_A = 0°C to 70°C; V_{CC} = +5V ± 10%; GND = 0V)

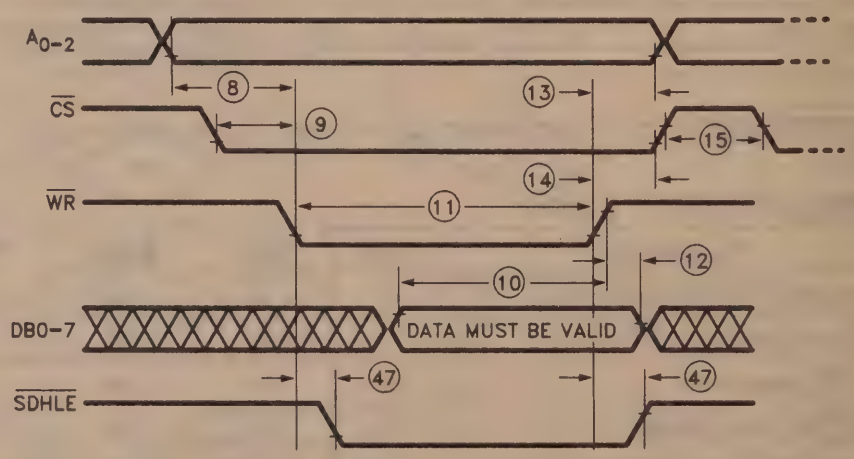
HOST READ TIMING WR CLOCK = 5.0 MHz

Symbol	Parameter	Min	Max	Units	Test Conditions
1	Address Stable Before $\overline{RD}$ ↓	0		ns	
2	Data Delay from $\overline{RD}$ ↓		70	ns	
3	$\overline{RD}$ Pulse Width	0.2	10	μs	
4	$\overline{RD}$ to Data Floating	10	200	ns	
5	Address Hold Time after $\overline{RD}$ ↑	0		ns	
6	Read Recovery Time	300		ns	
7	$\overline{CS}$ Stable before $\overline{RD}$ ↓	0		ns	See Note 6



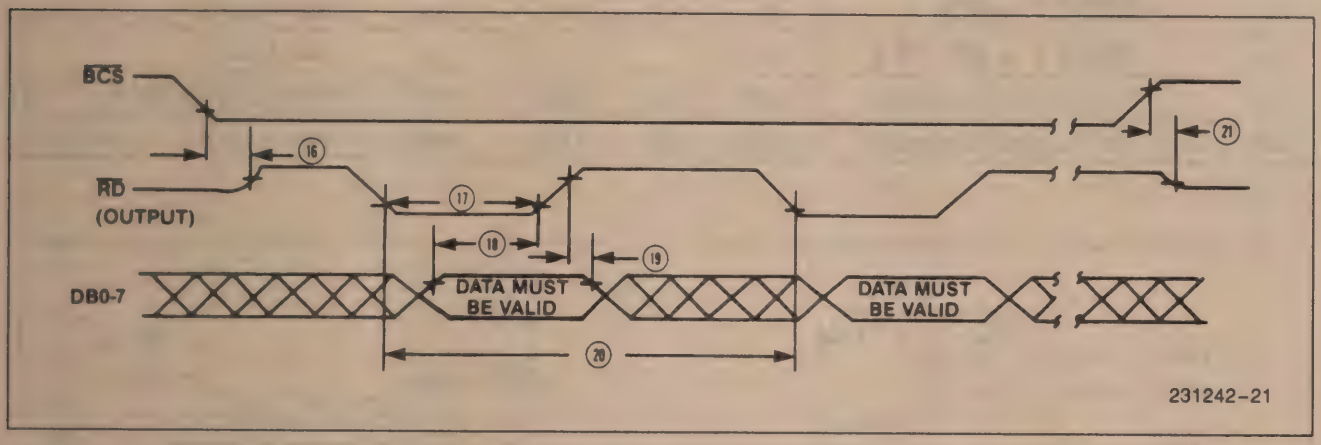
HOST WRITE TIMING WR CLOCK = 5.0 MHz

Symbol	Parameter	Min	Max	Units	Test Conditions
8	Address Stable Before $\overline{WR}$ ↓	0	10	μs	
9	$\overline{CS}$ Stable Before $\overline{WR}$ ↓	0	10	μs	
10	Data Setup Time Before $\overline{WR}$ ↑	0.16	10	μs	
11	$\overline{WR}$ Pulse Width	0.2	10	μs	
12	Data Hold Time After $\overline{WR}$ ↑	0		ns	
13	Address Hold Time After $\overline{WR}$ ↑	0		ns	
14	$\overline{CS}$ Hold Time After $\overline{WR}$ ↑	0		ns	See Note 7
15	Write Recovery Time	300		ns	
47	$\overline{SDHLE}$ Propagation Delay	20	150	ns	



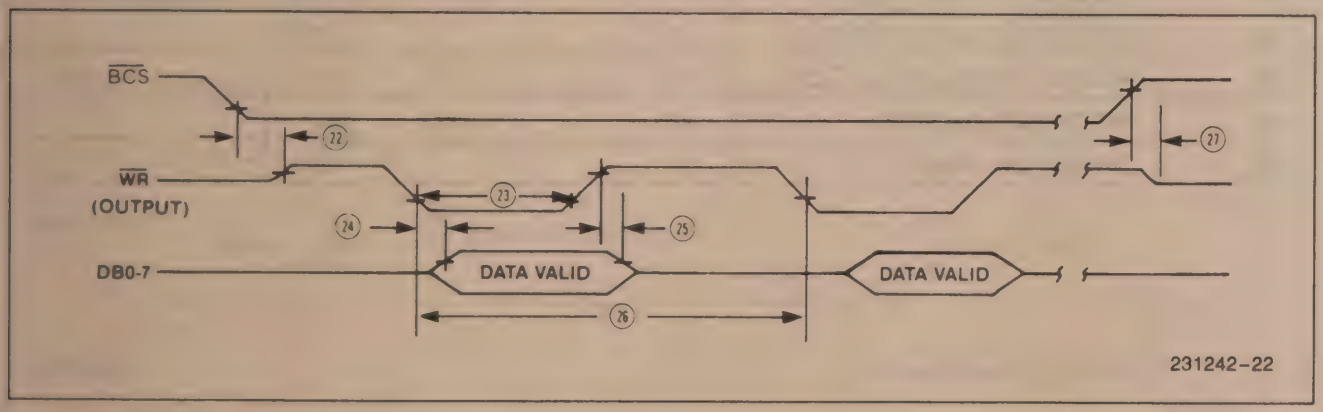
BUFFER READ TIMING (WRITE SECTOR COMMAND) WR CLOCK = 5.0 MHz

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
16	$\overline{BCS}$ ↓ to $\overline{RD}$ Valid	0		100	ns	
17	$\overline{RD}$ Output Pulse Width	300	400	500	ns	See Note 3
18	Data Setup to $\overline{RD}$ ↑	140			ns	
19	Data Hold from $\overline{RD}$ ↑	0			ns	
20	$\overline{RD}$ Repetition Rate	1.2	1.6	2.0	μs	See Note 1
21	$\overline{RD}$ Float from $\overline{BCS}$ ↑	0		100	ns	



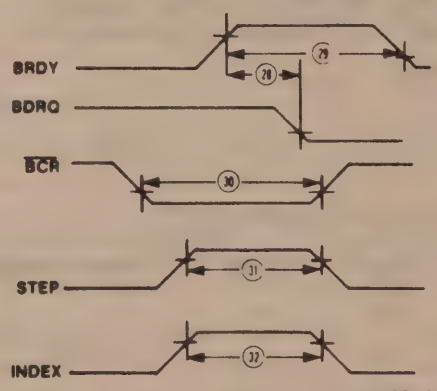
BUFFER WRITE TIMING (READ SECTOR COMMAND) WR CLOCK = 5.0 MHz

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
22	$\overline{BCS}$ ↓ to $\overline{WR}$ Valid	0		100	ns	
23	$\overline{WR}$ Output Pulse Width	300	400	500	ns	See Note 3
24	Data Valid from $\overline{WR}$ ↓			150	ns	
25	Data Hold from $\overline{WR}$ ↑	60		200	ns	
26	$\overline{WR}$ Repetition Rate	1.2	1.6	2.0	μs	See Note 1
27	$\overline{WR}$ Float from $\overline{BCS}$ ↑	0		100	ns	

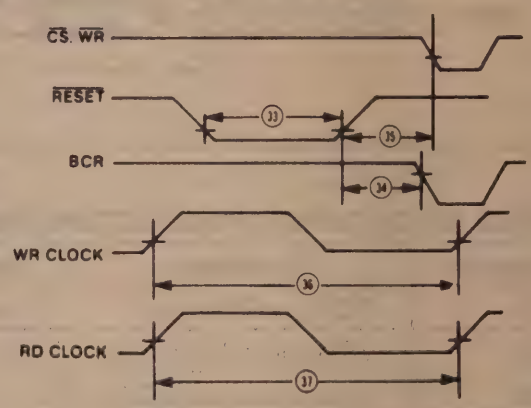


MISCELLANEOUS TIMING

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
28	BDRQ Reset from BRDY	20		200	ns	
29	BRDY Pulse Width	400			ns	See Note 4
30	$\overline{BCR}$ Pulse Width	1.4	1.6	1.8	μs	See Note 1
31	STEP Pulse Width	1.5	1.6	1.7	μs	Step Rate = 3.2 μs/step
		7.9	8.4	8.7	μs	All other step rates
32	INDEX Pulse Width	500			ns	
33	$\overline{RESET}$ Pulse Width	24			WR CLK	See Note 2
34	$\overline{RESET}$ ↑ to $\overline{BCR}$	0	3.2	6.4	μs	See Note 1
35	$\overline{RESET}$ ↑ to $\overline{WR}$, $\overline{CS}$ ↓	6.4			μs	See Note 1
36	WR CLOCK Frequency	0.25	5.0	5.25	MHz	50% Duty Cycle
37	RD CLOCK Frequency	0.25	5.0	5.25	MHz	See Note 5



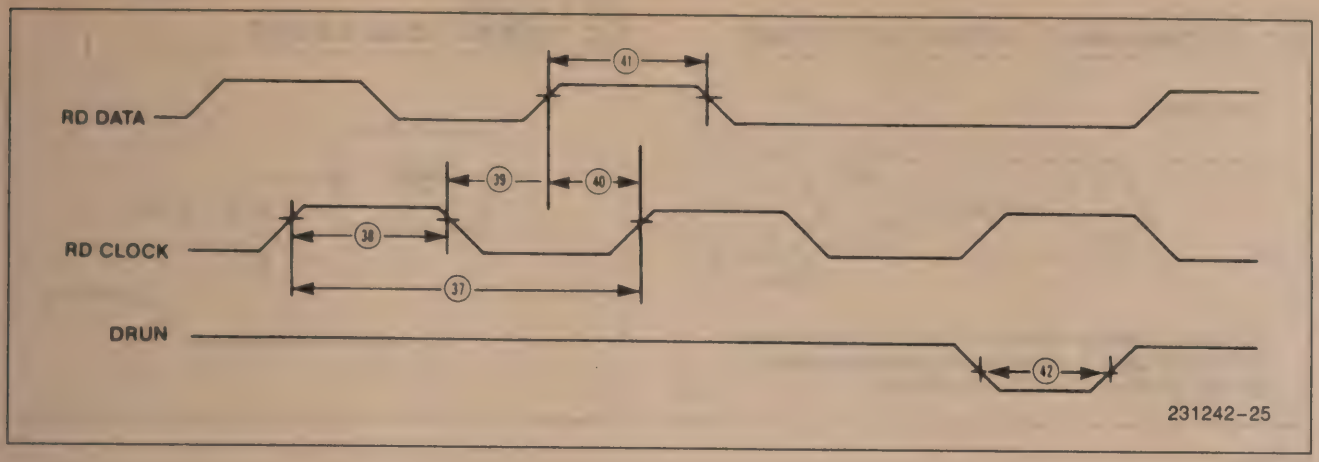
231242-23



231242-24

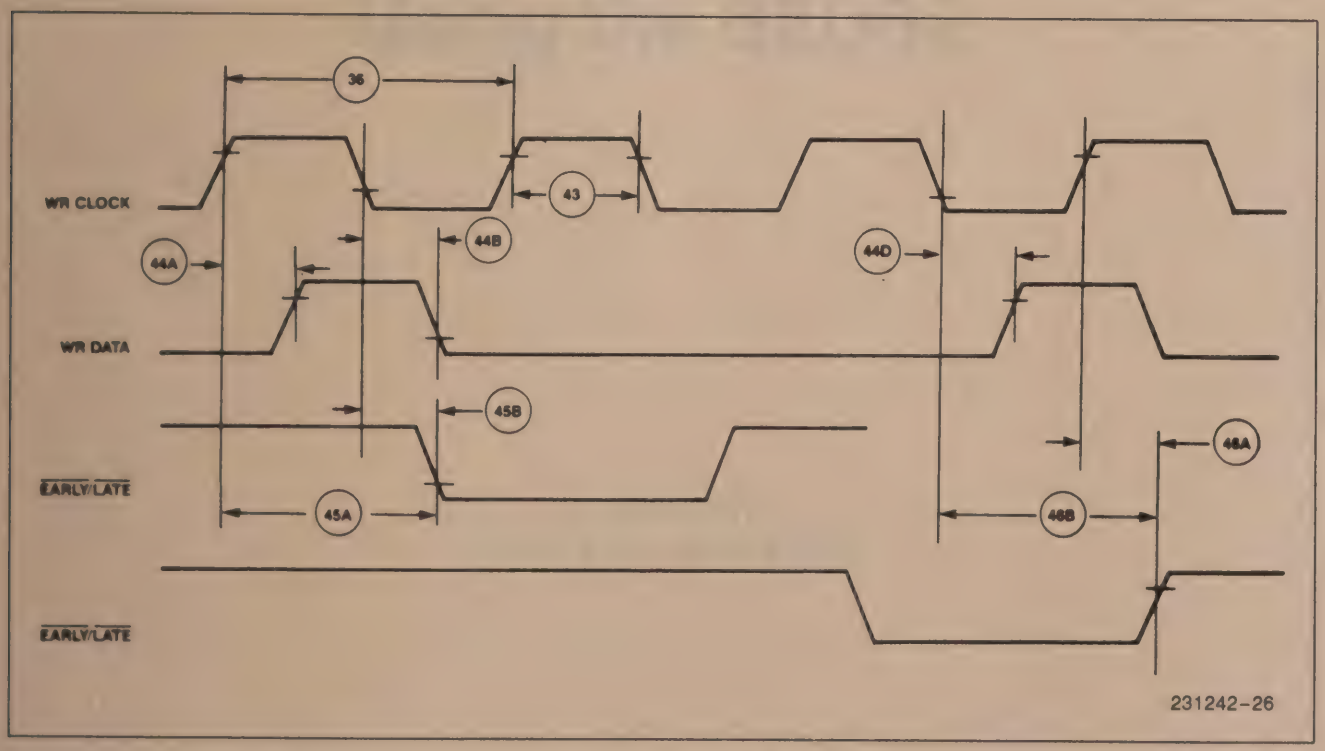
READ DATA TIMING WR CLOCK = 5.0 MHZ

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
38	RD CLOCK Pulse Width	95		2000	ns	50% Duty Cycle
39	RD DATA after RD CLOCK ↓	10			ns	
40	RD DATA before RD CLOCK ↑	20			ns	
41	RD DATA Pulse Width	40		T38/2	ns	
42	DRUN Pulse Width	30			ns	

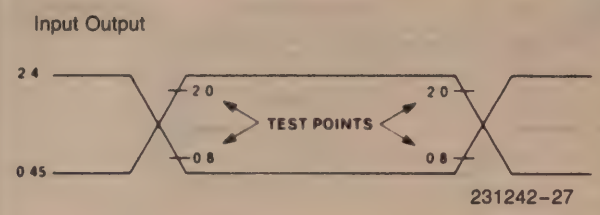


WRITE DATA TIMING WR CLOCK = 5.0 MHZ

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
43	WR CLOCK Pulse Width	95		2000	ns	50% Duty Cycle
	Propagation Delay					
44A	WR CLOCK ↑ to WR DATA ↑	10		65	ns	
44B	WR CLOCK ↓ to WR DATA ↓					
44D	WR CLOCK ↓ to WR DATA ↑					
45A	WR CLOCK ↑ to $\overline{\text{EARLY}}/\overline{\text{LATE}}$ ↓	10		65	ns	
45B	WR CLOCK ↓ to $\overline{\text{EARLY}}/\overline{\text{LATE}}$ ↓					
46A	WR CLOCK ↑ to $\overline{\text{EARLY}}/\overline{\text{LATE}}$ ↑	10		65	ns	
46B	WR CLOCK ↓ to $\overline{\text{EARLY}}/\overline{\text{LATE}}$ ↑					

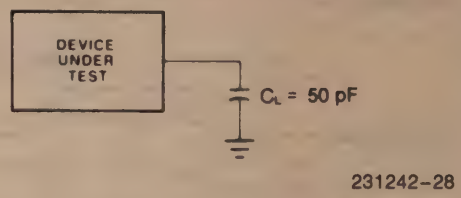


A.C. TESTING INPUT, OUTPUT WAVEFORM



AC Testing: Inputs Are Driven At 2.4V For A Logic .1, And 0.45V For A Logic .0. Timing Measurements Are Made At 2.0V For a Logic .1, And 0.8V For A Logic .0.

A.C. TESTING LOAD CIRCUIT



C_L Includes Jig Capacitance

NOTES

1. Based on WR CLOCK = 5.0 MHz
2. 24 WR CLOCK periods = 4.8 μs at 5.0 MHz.
3. 2 WR CLOCK periods ± 100 ns.
4. Previous restrictions on BRDY no longer apply. There are no restrictions on when BRDY may come. BRDY may be connected directly to BDRQ.
5. WR CLOCK Frequency = RD CLOCK Frequency ± 15%.
6. $\overline{RD}$ may be asserted before $\overline{CS}$ as long as it remains active for at least the minimum. T3 pulse width after $\overline{CS}$ is asserted.
7. $\overline{WR}$ may be asserted before $\overline{CS}$ as long as it remains active for at least the minimum T11 pulse width after $\overline{CS}$ is asserted.

intel

APPLICATION NOTE

AP-182

July 1984

Multimodule™ Winchester Controller Using the 82062

J. SLEEZER
TECHNICAL MARKETING

© INTEL CORPORATION, 1984

Order Number: 231133-002

MULTIMODULE™ WINCHESTER CONTROLLER USING THE 82062

CONTENTS

INTRODUCTION
ST506 Winchester Drive Overview
82062 WINCHESTER DISK CONTROLLER
Clock Inputs
Microprocessor Interface
Sector Buffer Control
Data Transfer Logic
Drive Interface
Microprocessor Interfaces
PIN DESCRIPTIONS
TASK REGISTER FILE
Error Register
Reduce Write Current Register
Sector Count Register
Sector Number
Cylinder Number Low Register
Cylinder Number High Register
Sector/Drive/Head Register
Status Register
Command Register
PROGRAMMING THE 82062
Commands
Software Section: General Programming
APPLICATION EXAMPLE
iSBX Bus Multimodule Boards
The SBX82062 Design Example
Software Driver Overview

CONTENTS

APPENDIX A

ST506 INTERFACE

THE ST506 INTERFACE

Data Transfer Rate

ID Fields

Sector Interleaving

Electrical Interface

ST412 HP (High Performance) Interface

CONTENTS

APPENDIX B

SOFTWARE DRIVER

APPENDIX C

SCHEMATICS

APPENDIX D

PAL SCHEMATICS

INTRODUCTION

The 82062 Winchester Disk Controller (WDC) was developed to ease the complex task of interfacing Winchester disk drives to microprocessor systems. Specifically, the 82062 WDC interfaces to drives that conform to the ST506 specification, which is the dominant interface for 5¼ inch drives. This Application Note provides some background on the 82062 WDC, the drive interfaces and general software routines. It concludes with a design example using the 82062 WDC interfaced to the SBX™ bus. Appendix B contains the listing of the software necessary to operate this controller board.

ST506 Winchester Drive Overview

Since the 82062 WDC interfaces only to drives conforming to the ST506 specification, this overview will limit itself to those drives. A summary of the ST506 specification is shown in Appendix A for those who are not familiar with it. The ST506 Winchester Disk contains from 1 to 8 hard disks (or platters) with the average being 2 to 3 disks. These disks are made from aluminum (hence the term hard disk) and are coated with some type of recording media. The recording media is typically made of magnetic-oxide, which is similar to the material used on floppy disks and cassette tapes. Each side of a hard disk is coated with recording media and each side can store data. Each surface of a disk has its own read/write head.

Hard disk drives are sealed units because the R/W heads actually fly above the disk surface at about 8 to 20 microinches. A piece of dust or dirt, which appears as a boulder to the gap between the heads and the disk surface, will wreak havoc upon the disk media.

The R/W heads are mechanically connected together and move as a single unit across the surface of the disk. There are 2 basic methods for positioning the heads. The first is with stepper motors, which is the most common method and is also used on most floppy disk drives. These positioners are used mainly because of their low cost.

The second method of positioning the heads is to use a voice-coil mechanism. These units do not move in steps but swing across the disk. These mechanisms generally permit greater track density than steppers, but also require complex feedback electronics which increases the cost of the drive. Generally, voice-coil head positioners use closed loop servo positioning, as compared to the open loop positioning used with stepper motors.

The surface of a disk is divided logically into concentric circles radiating from the center as shown in Figure 1. Each concentric circle is called a track.

The group of same tracks on all cylinders is collectively called a cylinder. The number of tracks on a surface (which affects storage density) is determined by the head positioners. Typically, stepper head positioners have fewer tracks than drives that use a voice coil positioner. Which type of positioner is used is irrelevant to the 82062 as positioners are part of the drive electronics. The 82062 can access up to 1024 tracks per surface.

Once the surface is divided into cylinders it is further divided radially (as with a pie). The area between the radial spokes is referred to as a sector. The number of sectors per track is determined by many variables, but is basically determined by the number of data bytes and the length of the ID field (which locates a sector). Figure 2 shows one manufacturer's specifications for their drive. The manufacturer formats the drive with 32–256 byte sectors per track. Alternatively, the drive could be reformatted to contain 17–512. byte sectors per track. This second option has fewer sectors per track but stores more data. Determining how many bytes each sector contains is done by extensive analysis of the hardware and operating system. The 82062 WDC is programmable for sector size during formatting.

The order in which sectors are logically numbered on the track is called interleaving. An interleave factor of four would have three sectors separating logically se-

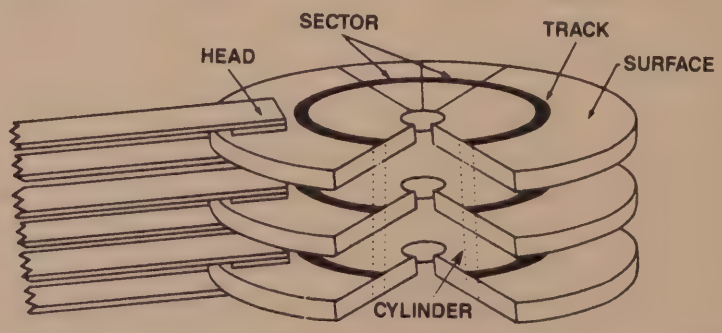


Figure 1

Capacity	
Unformatted	
Per Drive	6.38 Megabytes
Per Surface	1.59 Megabytes
Per Track	10416 Bytes
Formatted	
Per Drive	5.0 Megabytes
Per Surface	1.25 Megabytes
Per Track	8192 Bytes
Per Sector	256 Bytes
Sectors per Track	32
Transfer Rate	5.0 Megabits per second
Access Time	
Track to Track	3 ms
Average (Inc. Settle)	170 ms
Maximum (Inc. Settle)	500 ms
Settling Time	15 ms
Average Latency	8.33 ms
Functional Specifications	
Rotational speed	3600 rpm ± 1%
Recording density	7690 bpi max
Flux density	7690 fci
Track density	255 tpi
Cylinders	153
Tracks	612
R/W Heads	4
Disks	2

Figure 2. A Typical Drive Specification

quential sectors. Starting at the index pulse, an example of four way interleaving is:

Sector 1, Sector X, Sector Y, Sector Z, Sector 2, Sector . . .

Interleaving is used primarily because one sector at a time is transferred from disk to sector buffer to system RAM. Without interleaving, the delay in transferring data would result in sectors on the disk rotating past the heads. The operating system would then have to wait one disk revolution to get to the next sector (a 16.7 msec delay). With interleaved sectors, the next logical sector would be positioned beneath the heads after the previous sector of data had been transferred to the system RAM. Interleaving unfortunately slows down the overall transfer rate from the disk. A 5 Mbit/second transfer rate averages out to a 1.25 Mbit/second transfer rate when many sectors are transferred with four way interleaving. Again, how much interleaving to use is determined by extensive hardware/software benchmarking.

Whenever data is stored on a multiple platter disk drive, the same track on all surfaces whould be used before repositioning the heads to another track. Repositioning the heads generates a longer delay due to the mechanical delay of moving the heads. Switching to another head incurs no mechanical positioning delay. Only one head can be selected at a time.

Hard disk drives tend to be faster than floppies for two reasons. The speed at which the disk spins is about 10 times faster than the floppy (a floppy spins at 360 rpm). This yields an immediate one-tenth reduction in access times for the same size drive. While both ST506 drives and floppies use stepper motors, the steppers utilized by the hard disk drives are approximately twice as fast as those used by floppies.

82062 WINCHESTER DISK CONTROLLER

The 82062 WDC provides most of the functions necessary to interface between a microprocessor and an ST506 compatible disk drive. The 82062 converts the high level commands and parallel data of a microprocessor bus into ST506 compatible disk control signals and serial MFM encoded data. This section presents a detailed description of the 82062 and a discussion of various techniques which can be used to interface the 82062 to a microprocessor.

The internal structure of the 82062 is divided into several sections as shown in Figure 3. They are:

1. the microprocessor interface which includes the status and task registers;
2. sector buffer control;
3. the drive interface;
4. and the data transfer section, which includes the CRC logic and the conversion and MFM encoding/decoding of microprocessor data.

Clock Inputs

The 82062 has two clock inputs: read clock (RD CLOCK) and write clock (WR CLOCK). The PLA controller, the processor interface, buffer control and MFM encoding sections operate off the WR CLOCK input. The RD CLOCK input is used only for decoding the MFM data stream. The clocks may be asynchronous to one another. Both clocks have non-TTL compatible inputs. The easiest method to interface to TTL requires a pull-up resistor to satisfy their input voltage needs. The resistor's value must be compatible with the VIL specification of these pins. See the Pin Descriptions Section for more specific information.

Microprocessor Interface

The microprocessor interface of the 82062 contains the control logic which permits commands and data to be

or when data needs to be transferred to the host from the sector buffer. This is discussed further in the Interrupt Mode Section. When selecting the data transfer option, the interrupt line will go active at the same time as the BDRQ line and the interrupt will be removed only when the proper handshake occurs with the sector buffer.

Task Registers

The Task Register File contains the command, status, track number, sector number, and other information necessary to properly execute a command. These registers are accessed with A0–A2, $\overline{RD}$ (or $\overline{WR}$), and $\overline{CS}$ being valid and are not cleared by a reset. The registers are covered in detail in the Task Register File Section.

Sector Buffer Control

The 82062 was designed to operate with an external buffer equal in size to one sector. To ease the design-in of this buffer, the 82062 provides all of the control signals it needs to operate the buffer. This buffer must be isolated from the system bus, using tri-state buffers, during disk transfers to prevent contention during the period that the 82062 is accessing the buffer. A sector buffer is generally used to ease interfacing to the system due to the high disk data rates (625 kbytes/sec), although it is not required.

$\overline{BCS}$

The Buffer Chip Select ($\overline{BCS}$) line goes active whenever the 82062 is accessing the sector buffer. This signal should remove the microprocessors ability to access the 82062 and sector buffer and must enable the sector buffer for use by the 82062.

At a 5 Mbit/sec disk data rate, the 82062 will access the buffer every 1.6 microseconds (8 bits × 200 ns/bit). $\overline{BCS}$ will remain low the entire time the 82062 is accessing the buffer. The 82062 will pulse the appropriate $\overline{RD}$ or $\overline{WR}$ line for each byte transferred.

$\overline{BCR}$

Buffer Counter Reset ($\overline{BCR}$) goes active each time that $\overline{BCS}$ changes state. Its purpose is to reset the address counter of the sector buffer back to zero before and after the 82062 uses the sector buffer. Its function is optimized for single sector transfers. Multiple sector transfers should use a software controlled buffer counter reset and not use $\overline{BCR}$ as the sector buffer will be reset to the beginning after each sector is transferred.

BDRQ, BRDY

Buffer Data Request (BDRQ) and Buffer Ready (BRDY) provide the handshake needed to transfer data between the sector buffer and the host. BDRQ signals that data must be moved to/from the sector buffer and the host. BRDY has two functions. Once the transfer signaled by BDRQ is finished, asserting BRDY will inform the 82062 that the transfer is completed and that it may finish executing the command. BRDY is also used in multiple sector commands. BRDY going high during a multiple sector transfer indicates that the buffer is full (or empty—depending upon the command) and the transfer should wait until the buffer is serviced. The sector that was being transferred will finish and the 82062 will deactivate $\overline{BCS}$ and activate BDRQ. The host microprocessor must then transfer the data between the buffer and system memory. When this transfer is finished, asserting BRDY will cause the 82062 to resume the command.

The handshaking between BDRQ and BRDY occurs only in full sector increments—not on a byte basis. A high on BDRQ indicates a full sector's worth of data is required; BRDY going high indicates a full sector of data is available to the 82062 without interruption.

Only the rising edge of BRDY is valid. A falling edge may occur at any time without effect. $\overline{BCR}$ will pulse and $\overline{BCS}$ will go active eight byte times (8 bytes × 8 bits/byte × 200 ns/bit = 12.8 microseconds) before the first data byte is transferred from the sector buffer to the disk.

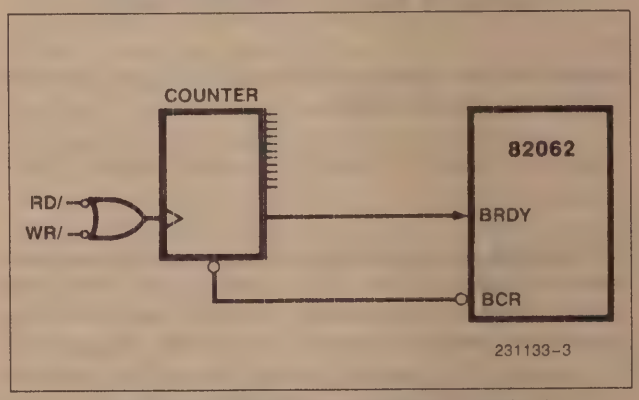


Figure 4. BRDY Generation Logic

Data Transfer Logic

This section of the 82062 is responsible for conversion of serial disk data to parallel data (and vice versa); encoding/decoding of the disk's MFM serial bit stream; detecting the address mark; and verifying data integrity through its CRC generation and checking logic.

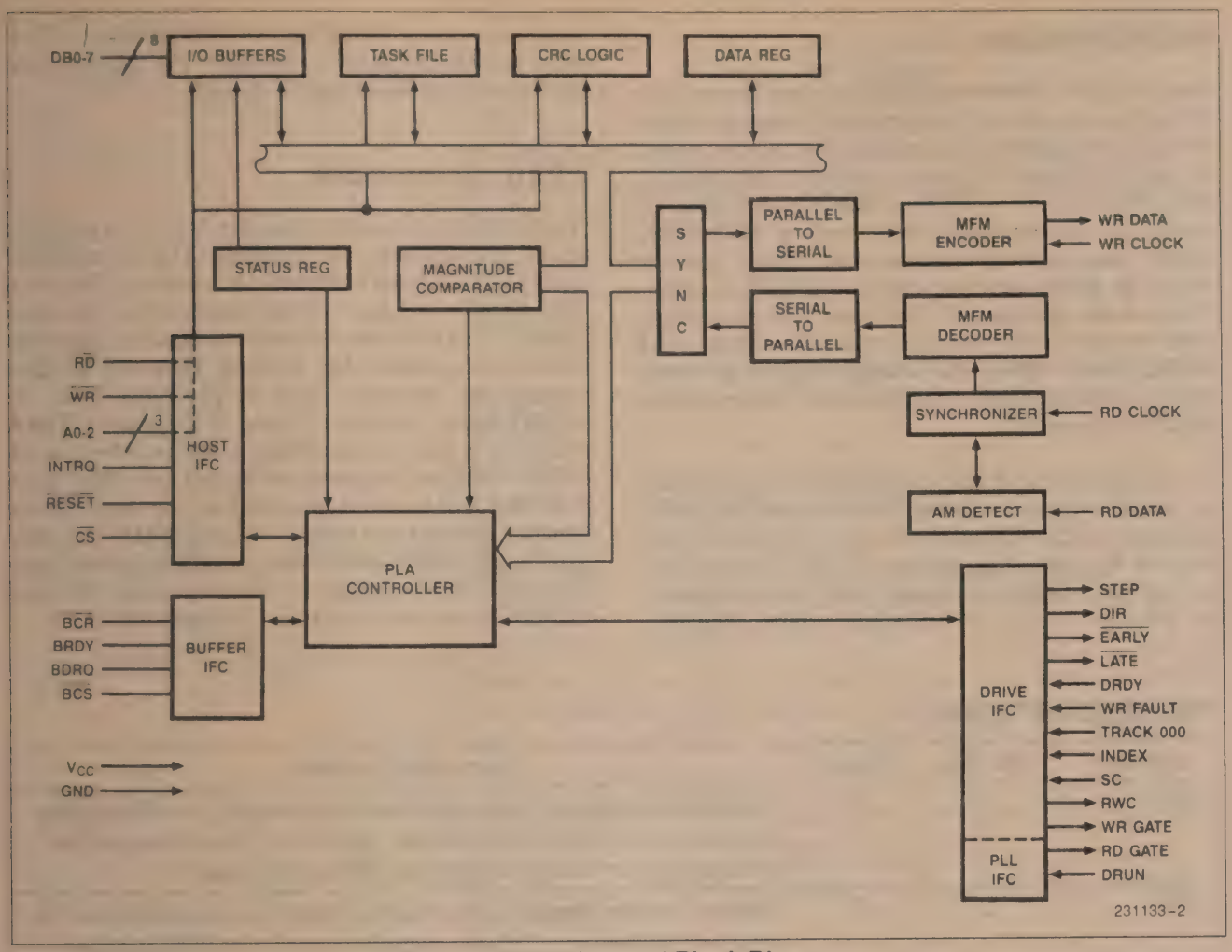


Figure 3. 82062 Internal Block Diagram

transferred between the host and the 82062. The interface consists of an 8 bit, tri-state, bidirectional data bus; the task registers; a 3 to 8 address decoder for selecting one of the seven registers; and the general read, write, and chip select logic. Externally, the 82062 expects a buffer equal in size to a sector on the disk, and tri-state transceivers between the sector buffer and the microprocessors data bus in order to isolate itself from the microprocessor during disk data transfers.

A0–A2, Data Bus

These three address lines are active high signals and select one of the seven register locations in the 82062. They are not latched internally. If the three addresses are equal to 0 and the 82062 is selected, the data bus is kept tri-stated to ease interfacing to a sector buffer. The 82062's data bus is controlled by both the microprocessor and the 82062. The microprocessor has control for loading the registers and command. During disk reads or writes, control switches to the 82062 so that it may access the local sector buffer when transferring data between the disk and the buffer.

$\overline{RD}$, $\overline{WR}$, $\overline{CS}$

The chip select ($\overline{CS}$) is typically decoded from the higher order address lines. $\overline{CS}$ only permits data to be placed into, or read from, the 82062's task registers. Once a disk operation starts, $\overline{CS}$ no longer efffects the 82062. $\overline{RD}$ and $\overline{WR}$ are bidirectional lines and are used to read or write the 82062's registers by the host microprocessor and are valid only if $\overline{CS}$ is present. The 82062 will drive $\overline{RD}$ and $\overline{WR}$ when transferring data between the sector buffer and the disk. A signal is provided to tri-state the $\overline{RD}$ and $\overline{WR}$ lines from the host during a buffer access. This is covered in the Sector Buffer Control Section.

Interrupts

An interrupt is issued at the end of all commands, and the interrupt is cleared by reading any register. For the Read Sector command only, the 82062 allows the user the option of an interrupt either at the termination of the command, as is the case with all other commands,

Polled Interface

Since the 82062 isolates itself from the host during several commands, the host cannot read the status register during some periods to determine what course should be taken. In Figure 10, trying to read the status register when $\overline{BCS}$ is active will return indeterminate data. To prevent the microprocessor from reading this indeterminate data, a hardware generated "Busy" pattern should be driven onto the data bus if $\overline{BCS}$ is active. This is shown in Figure 11. The status register contains a data request (DRQ) bit whose timing is equal to the BDRQ output signal, thus making a polled operation possible. DRQ will stay set in the status register until a BRDY is generated.

One design issue with the polled interface occurs when the microprocessor is polling the status and the 82062 deactivates $\overline{BCS}$. The microprocessor would normally read the hardware busy pattern. If $\overline{BCS}$ is deasserted, the hardware pattern is disabled and the microprocessor will start to read the real status register. The read cycle may almost be finished, and the read access period of the 82062 will not be satisfied. The data returned to the microprocessor will be invalid.

Interrupt Interface

There are cases where the designer does not want to tie up the microprocessor with polling. The typical 82062 design will need two interrupts per command. One for a data transfer and one for the completion of the command. The 82062 has an output to issue an interrupt when the command has finished. However for data transfers an interrupt must be generated from the BDRQ line as shown in Figure 12 (whether a DMA controller is used or not). When a data transfer is needed, the 82062 will activate the BDRQ line. The microprocessor will be interrupted and do the data transfer function. BDRQ will stay active until BRDY is generated, so the system must either use edge triggered interrupts or must not write the end-of-interrupt byte until BDRQ is removed (this is true of Intel's 8259A).

PIN DESCRIPTIONS

Symbol	Pin. No.	Type	Name and Function
$\overline{BCS}$	1	O	**Buffer Chip Select:** Output used to enable reading or writing of the external sector buffer by the 82062. When low, the host should not be able to drive the 82062 data bus, $\overline{RD}$, or $\overline{WR}$ lines.
$\overline{BCR}$	2	O	**Buffer Counter Reset:** Output that is strobed by the 82062 prior to read/write operation. This pin is strobed whenever $\overline{BCS}$ changes state. Used to reset the address counter of the buffer memory.
INTRQ	3	O	**Interrupt Request:** Interrupt generated by the 82062 upon command termination. It is reset when any register is read. Optionally signifies when a data transfer is required on Read Sector commands.
N/C	4		No connection. Reserved for future use.
$\overline{RESET}$	5	I	**Reset:** Initializes the controller and clears all status flags. Does not clear the Task Registers.
$\overline{RD}$	6	I/O	**Read:** As an input, $\overline{RD}$ controls the transfer of information from the 82062 registers to the host. $\overline{RD}$ is an output when the 82062 is reading data from the sector buffer ($\overline{BCS}$ low).
$\overline{WR}$	7	I/O	**Write:** As an input, $\overline{WR}$ controls the transfer of command or task information into the 82062 registers. $\overline{WR}$ is an output when the 82062 is writing data to the sector buffer ($\overline{BCS}$ low).
$\overline{CS}$	8	I	**Chip Select:** Enables $\overline{RD}$ and $\overline{WR}$ as inputs for access to the Task Registers. It has no effect once a disk command starts.
A0–A2	9–11	I	**Address:** Used to select a register from the task register file.
DB0–DB7	12–19	I/O	**Data Bus:** Bidirectional 8-bit Data Bus with control determined by $\overline{BCS}$. When $\overline{BCS}$ is high the microprocessor has full control of the data bus for reading and writing the Task Registers. When $\overline{BCS}$ is low the 82062 controls the data bus to transfer data to or from the buffer.

Pin Descriptions (continued)

Symbol	Pin. No.	Type	Name and Function
GND	20		Ground.
WR DATA	21	O	**Write Data:** Open drain output that shifts out MFM data at a rate determined by Write Clock. Final stage requires an external flip-flop clock at 10 MHz. See note 1.
$\overline{\text{LATE}}$	22	O	**Late:** Open drain output used to derive a delay value for write precompensation. Valid when WR GATE is high. Active on all cylinders. See note 1.
$\overline{\text{EARLY}}$	23	O	**Early:** Open drain output used to derive a delay value for write precompensation. Valid when WR GATE is high. Active on all cylinders. See note 1.
WR GATE	24	O	**Write Gate:** High when write data is valid. WR GATE goes low if the WR FAULT input is active. This output is used by the drive to enable head write current.
WR CLOCK	25	I	**Write Clock:** Clock input used to derive the write data rate. Frequency −5 MHz for the ST506 interface, 4.34 MHz for the SA 1000 interface. See Note 2.
DIR	26	O	**Direction:** High level on this output tells the drive to move the head inward (increasing cylinder number). The state of this signal is determined by the 82062's internal comparison of actual cylinder location vs desired cylinder.
STEP	27	O	**Step:** Provides 8.4 microsecond pulses to move the drive head to another cylinder at a programmable frequency.
DRDY	28	I	**Drive Ready:** If DRDY from the drive goes low, the command will be terminated.
INDEX	29	I	**Index:** Signal from the drive indicating the beginning of a track. It is used by the 82062 during formatting, and for counting retries. Index is edge triggered. Only the rising edge is valid.
WR FAULT	30	I	**Write Fault:** An error input to the 82062 which indicates a fault condition at the drive. If WR FAULT from the drive goes high, the command will be terminated.
TRACK 000	31	I	**Track Zero:** Signal from the drive which indicates that the head is at the outermost cylinder. Used by the Restore command.
SC	32	I	**Seek Complete:** Signal from the drive indicating to the 82062 that the drive head has settled and that reads or writes can be made. SC is edge triggered. Only the rising edge is valid.
RWC	33	O	**Reduced Write Current:** Signal goes high for all cylinder numbers above the value programmed in the Write Precomp Cylinder register. It is used by the precompensation logic and by the drive to reduce the effects of bit shifting.
DRUN	34	I	**Data Run:** This signal informs the 82062 when a field of ones or zeros has been detected by an external one-shot. This indicates the beginning of an ID field. RD GATE is brought high when DRUN is sampled high for 16 clock periods. See Note 2.
BRDY	35	I	**Buffer Ready:** Input used to signal the controller that the buffer is ready for reading (full), or writing (empty), by the host μP. Only the rising edge indicates the condition.

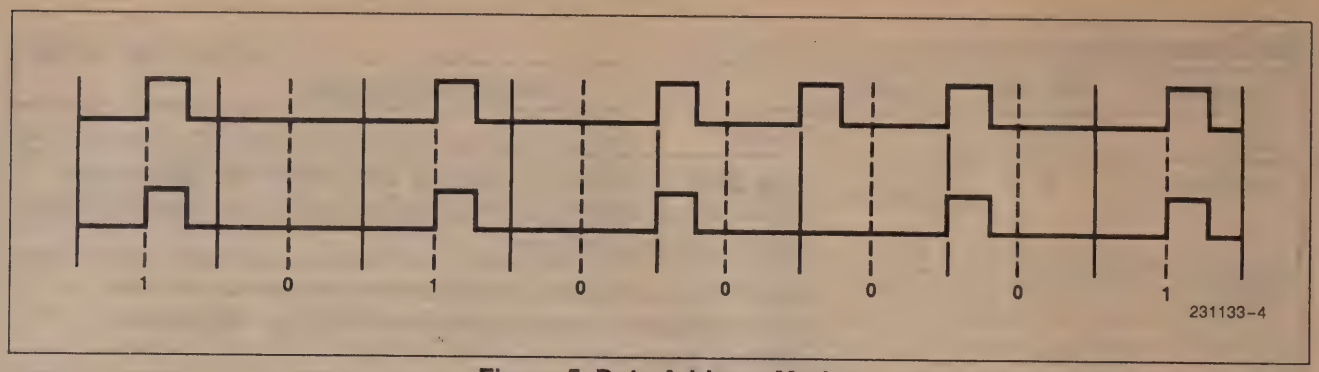


Figure 5. Data Address Mark

MFM Encoding/Decoding

The MFM encoding section will receive 8 bit parallel data when a valid command has been recognized and BRDY has gone high. The parallel data is first serialized and converted to an intermediate, NRZ encoded, data stream. The serial NRZ data is sent to the MFM encoding section and then transferred to the disk. Decoding of the MFM bit stream (during disk reads) happens in reverse order.

The control logic operates off the write clock (WR CLOCK) running at a frequency of the desired tranfer rate. The MFM decoding portion operates off of the read clock (RD CLOCK) input, which is supplied by an external phase lock loop. The two clocks need not be synchronized to each other. Data is written (and hence read) with the most significant bit first.

Address Mark Detector

The address mark is a unique 2 byte code written at the beginning of each ID field and data field. This address mark serves two purposes. It tells the controller what type of data is about to be received so that internal computations can be performed, and to ensure that ID fields are not sent to the host. The second purpose is to align the serial data back to the original 8 bit boundaries that existed when data was written (there are no byte boundaries on a disk).

An address mark is always preceded by the all zeros synchronization field. The 82062 starts comparing the incoming data stream when the synchronization field ends. A high speed comparator is used since the 82062 does not yet know where the proper byte boundaries are. When a proper comparison of the address mark is made the controller starts assembling bytes, starting with the second byte of the address mark.

The first byte of the address mark is an "A1" Hex, but purposely violates the MFM encoding rules by removing a clock pulse. In Figure 5, the first example is of a normal MFM encoded A1H. The second example is of the address mark and shows the missing clock pulse. The non-MFM compatible A1 is to prevent the host from issuing a similar data byte and possibly confusing detection logic.

The second byte specifies either an ID or data field and is encoded according to normal MFM rules. It is either an "F8" Hex for a data field, or "FC" through "FF" for an ID field. The different values correspond to a range of cylinders on the drive in increments of 256 tracks. The 82062 makes no use of this information, but writes it for compatibility with the ST506 specification during formatting.

CRC Generation/Checking

The CRC generator computes and checks the cyclic redundancy check bytes that are appended to the ID and data fields. CRC generation/checking is always done on ID fields. Data fields have a choice between 82062 CRC or externally supplied ECC. Read Sector commands with a CRC error will still have transferred the data into the sector buffer. When bit 7 in the SDH register is low (enabling CRC for data fields) the CRC bytes are not transferred to the sector buffer or host.

The generator polynomial for the CRC-CCITT (CRC-16) code is:

$$x16 + x12 + x5 + 1 = (x + 1)(x15 + x14 + x13 + x12 + x4 + x3 + x2 + x + 1)$$

The code's capability is as follows:

a) Detects all occurrences of an odd number of bits in error.

b) Detects all single, double, and triple bit errors if the record length (including check bits) is less than 32,767 bits.

c) Detects all single-burst errors of sixteen bits or less.

d) Detects 99.99695% of all possible 17 bit burst errors, and 99.99847% of all possible longer burst, assuming all errors are possible and equally probable.

The CRC code has some double-burst capability when used with short records (sectors). For a 256 byte sector the code will detect double-bursts as long as the total number of bits in error does not exceed 7.

PLA Control

The PLA Controller interprets command sent by the microprocessor. Its operation is synchronized to the WR CLOCK input. The PLA controller is started when a command is written into the command register. It generates control signals and operates in a handshake mode when communicating with the MFM decoding block.

Magnitude Comparator

A 10 bit magnitude comparator is used to calculate the direction and number of step pulses needed to move the head from the present cylinder position to the desired position. A separate high speed equivalence comparator is used to compare ID field bytes when searching for a sector ID field.

Drive Interface

The drive interface of the 82062 contains the logic that makes possible the storage and reliable recovery of data. This interface consists of the drive and head select logic, the disk control signals, and read and write data logic as shown in Figure 6. This section describes the external circuitry which is required to complete the 82062's drive interface.

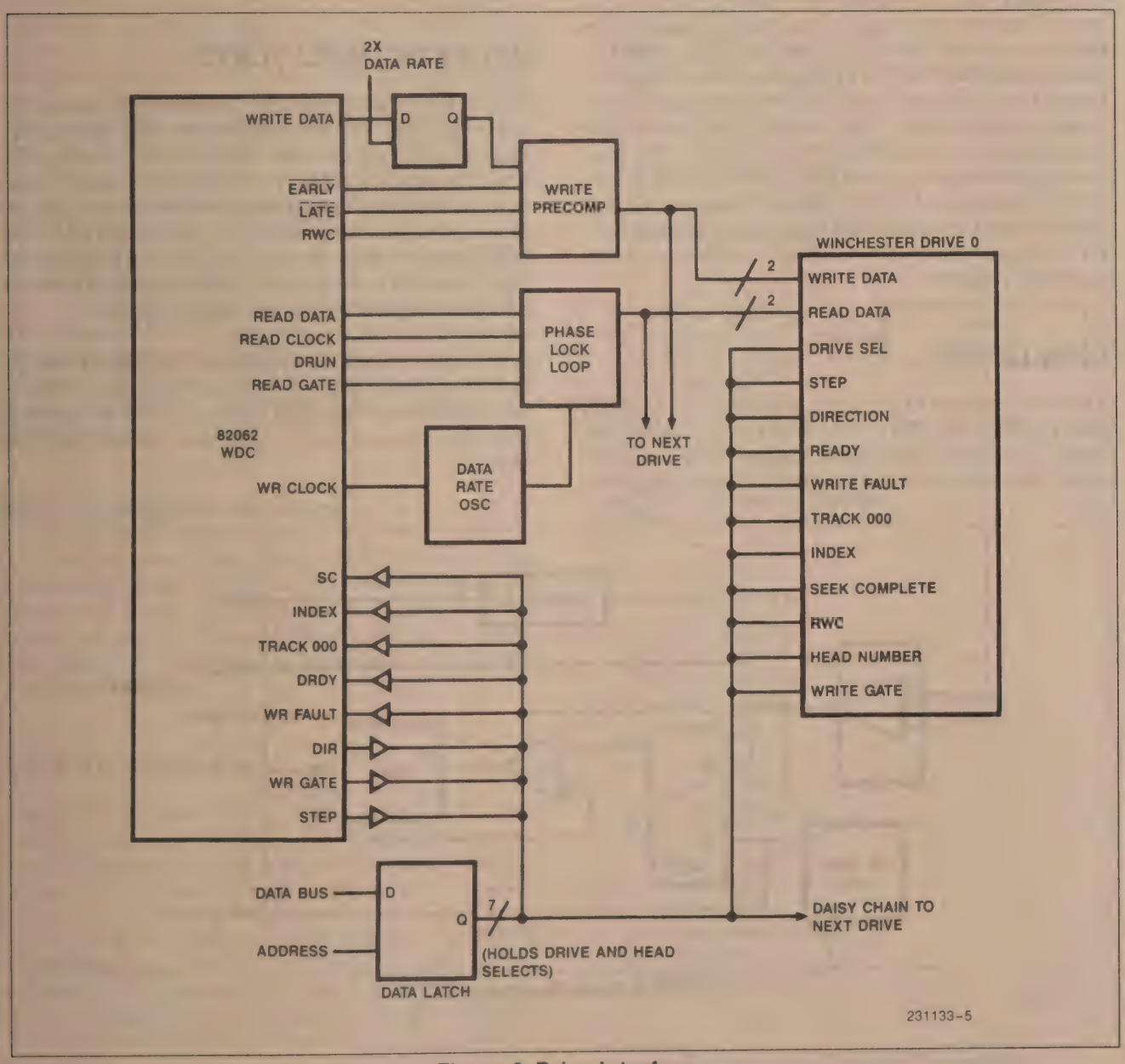


Figure 6. Drive Interface

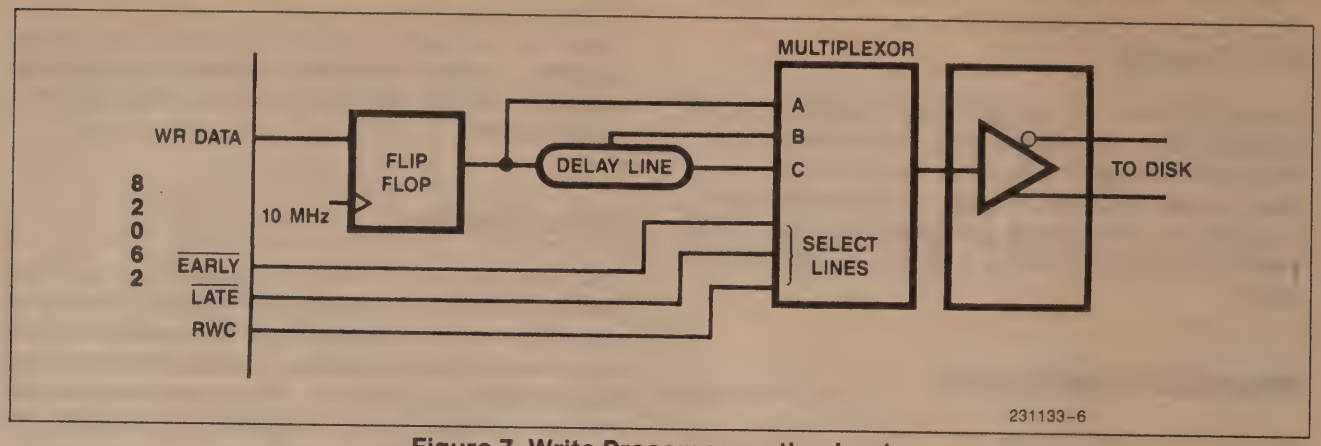


Figure 7. Write Precompensation Logic

Drive/Head Select

The 82062 has no outputs for selecting the head or drive. Therefore these signals must be generated by the user as shown in Figure 6. Data bits 0–4 should be latched whenever the SDH register is written. Bits 0–2 would then be driven onto the drive cable with open collector buffers. Bits 3 and 4 would be decoded after being latched, then buffered for the cable. The head information written to the 82062's SDH register is used to write the proper ID fields during formatting. Changing the drive bits in the SDH register will cause a Scan ID to be performed by the 82062 to update non user accessible registers.

Drive Control

The drive control (STEP, DIR, WR FAULT, TRACK 000, INDEX, SC, RWC, and WR GATE) signals are merely conditioned for transmission over the drive cable. The purpose of each pin can be found in the section on Pin Descriptions and their use in the Command Section.

WR DATA, $\overline{\text{EARLY}}$, $\overline{\text{LATE}}$

Figure 7 is a diagram of the interface required on the write data line. The final stage of the MFM encoding requires applying the WR DATA to an external flip-flop clocked at 10 MHz. The 82062 monitors the serial write data output for particular bit patterns that require precompensation to prevent bit shifting. $\overline{\text{EARLY}}$ and $\overline{\text{LATE}}$ are active on all cylinders and will normally require that RWC be factored into them to activate the data precompensation on the proper cylinder.

A delay line is required to generate the delayed data for precompensation since the actual delay varies between drive manufacturers. $\overline{\text{EARLY}}$ and $\overline{\text{LATE}}$ go active in the same clock period that generates the data bit to be shifted.

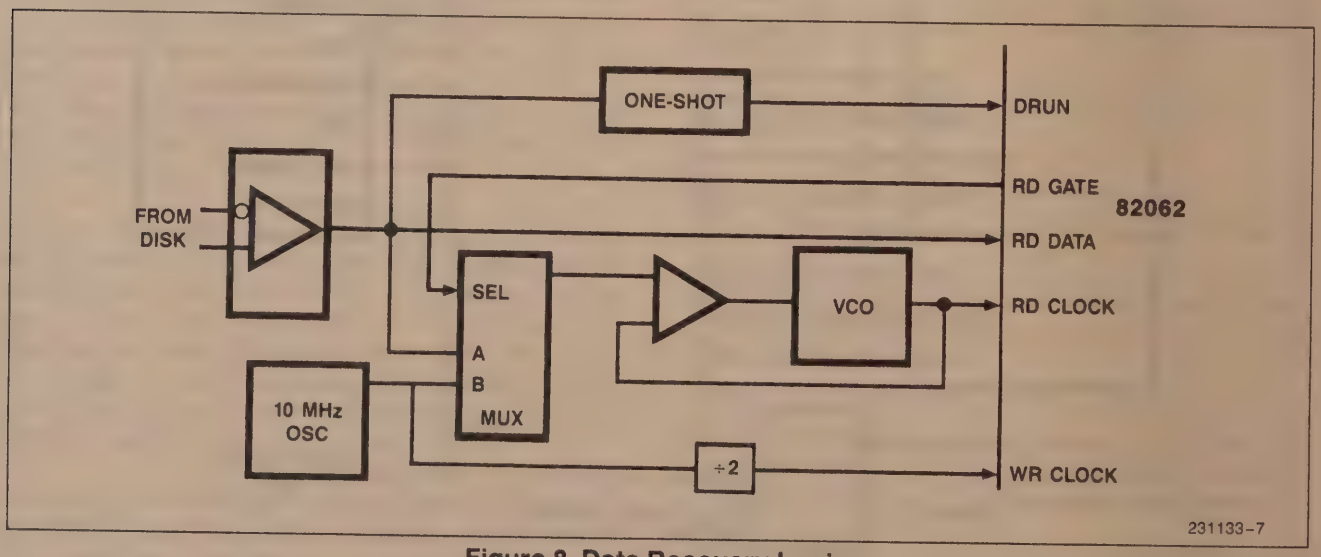


Figure 8. Data Recovery Logic

RD Data, DRUN, RD Gate

The read data interface is shown in Figure 8, and consists of the data run (DRUN) signal and a phase lock loop to generate the RD CLOCK input to decode the serial data. DRUN is generated from a retriggerable one-shot with a period just exceeding one bit cell. A sync field consisting of a string of clock pulses will continually retrigger the one-shot producing a steady high level on DRUN. The 82062 counts off 16 clock pulses internally, and if DRUN is still active, will make RD GATE active. Any byte other than an address mark will deactivate RD GATE and the sequence starts over.

The phase lock loop generates RD CLOCK which is used to decode the incoming serial data. Until RD GATE is activated by the 82062, the phase lock loop (PLL) should be locked onto a local 10 MHz clock to minimize PLL lock-up times. When RD GATE is activated, the PLL starts locking onto the incoming data stream, which should consist of the all zeros sync field. Once the PLL locks onto this synch field, the 82062 will start examining the serial data for a non-zero byte. A non-zero byte will be indicated by DRUN dropping since the address mark follows the sync field and is an "A1" Hex. This sequence is shown in Figure 9. If the address mark is detected, and if it was preceded by at least 9 bytes of zeros, RD GATE will stay active. The 82062 will then assemble bytes of data, and ensure the proper ID field is found. If a non-zero or non-address mark byte was detected, RD GATE will go inactive for a minimum of 2 byte times. If a data field or the wrong ID field is detected, or the ID field was not preceded by 8 bytes of zeros, then RD GATE goes inactive and the sequence starts over with the 82062 examining the DRUN input.

Microprocessor Interfaces

This section shows the general 82062 interfaces to a microprocessor system. There are essentially four interfaces which consist of a combination of polled, DMA, and interrupts. While the 82062 was designed to interface directly to one type, it accommodates all with minor additional logic.

DMA Interface

The 82062 is designed to use a DMA controller for data transfer between its sector buffer and the host system, and to interrupt the host when the command has finished. This interface is shown in Figure 10.

When the 82062 determines that a transfer is needed between the sector buffer and the host (either at the beginning of a command or through BRDY going active in a multiple sector transfer), it will assert BDRQ. BDRQ will initiate a DMA transfer via the DMA request input. The DMA controller will generate reads or writes which will increment an address counter. BRDY indicates that the data transfer has finished and is issued off the carry-out line (or high order address line) of the counter. The 82062 will assert BDRQ at this point and activate $\overline{BCS}$ to prevent the host from intefering with disk/buffer transfers. There can be no polling for a data transfer or a register read without an interrupt in this scheme.

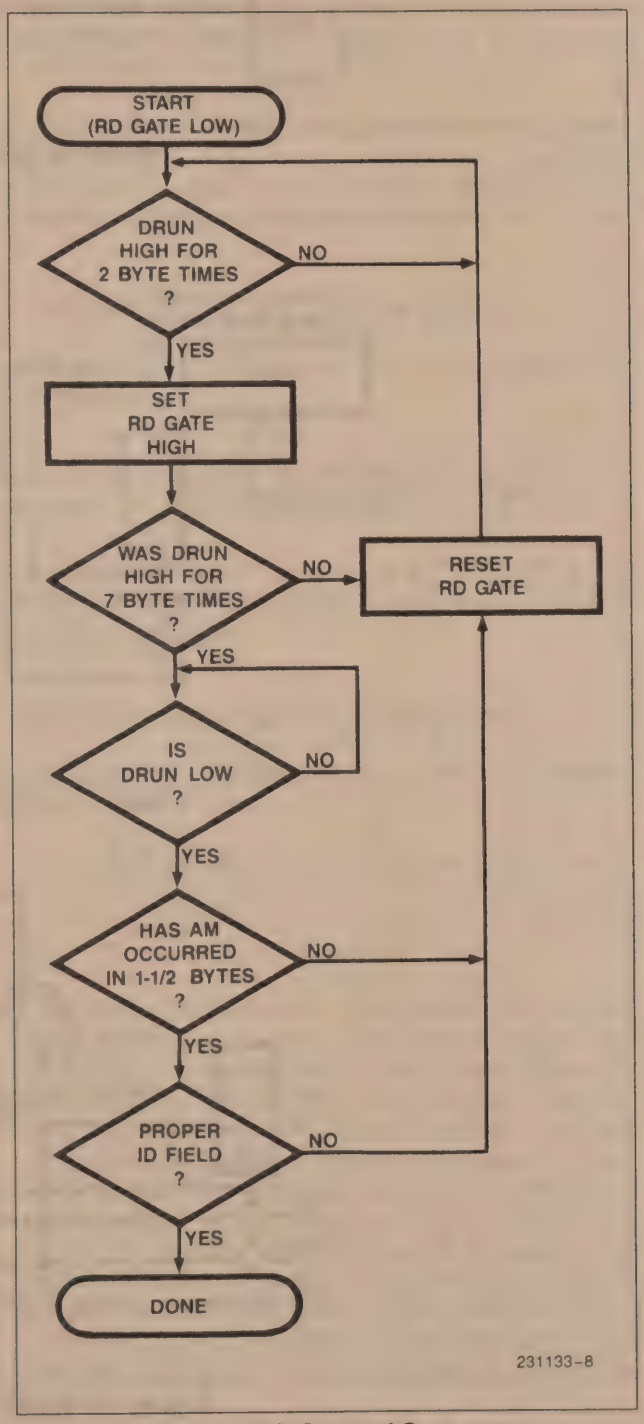


Figure 9. PLL Control Sequence

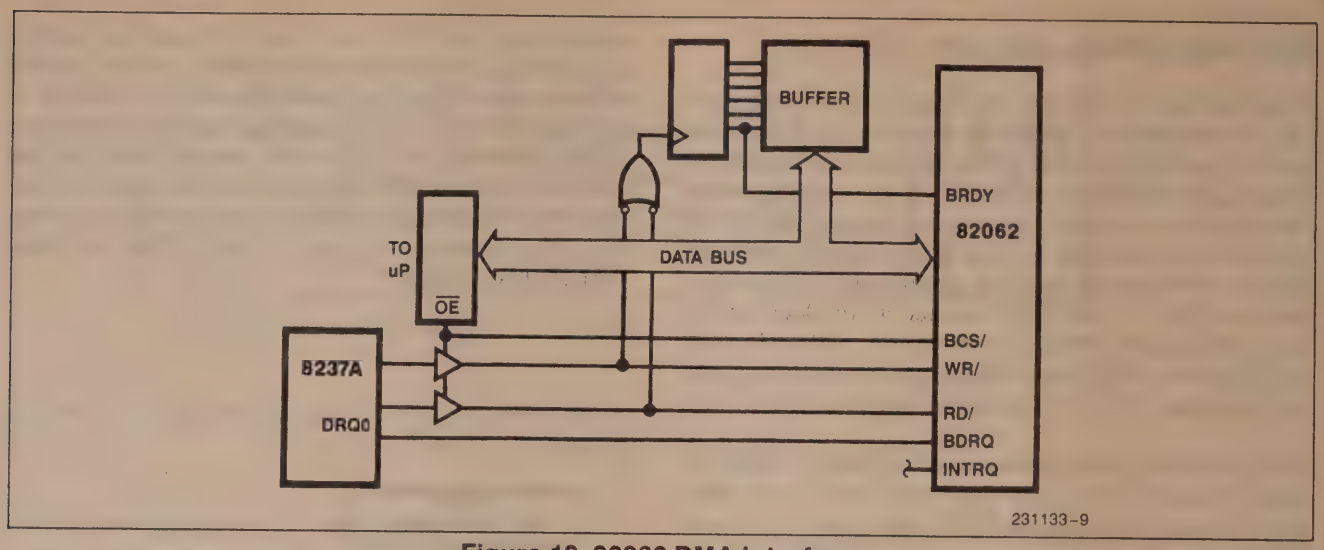


Figure 10. 82062 DMA Interface

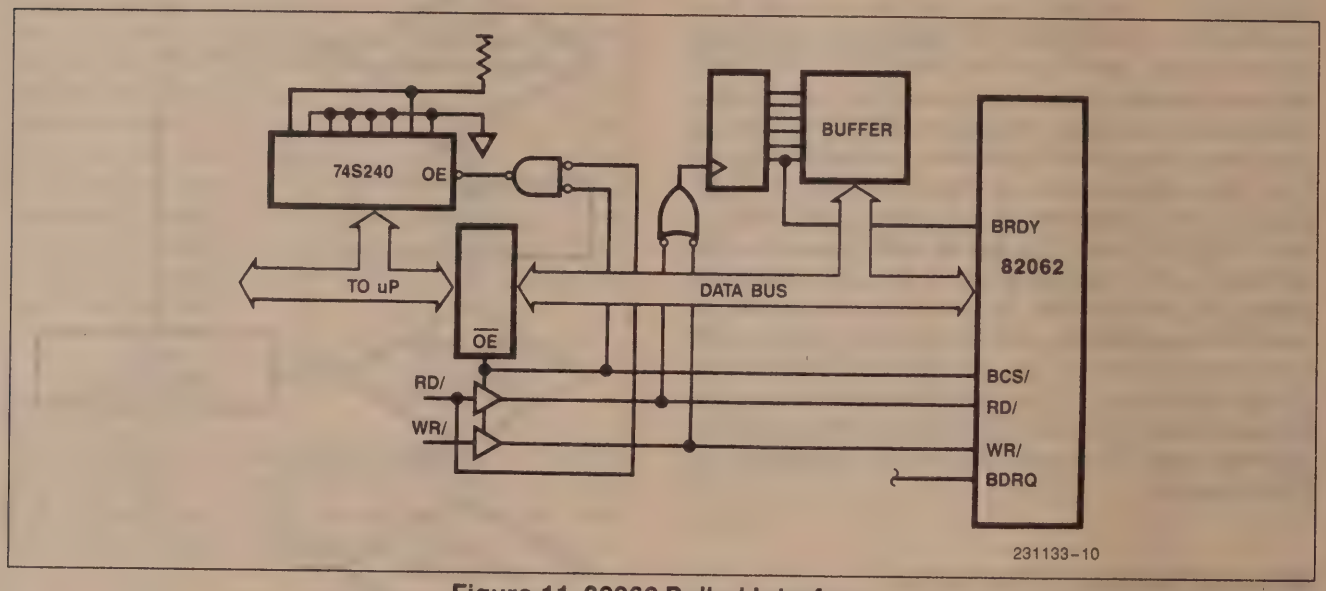


Figure 11. 82062 Polled Interface

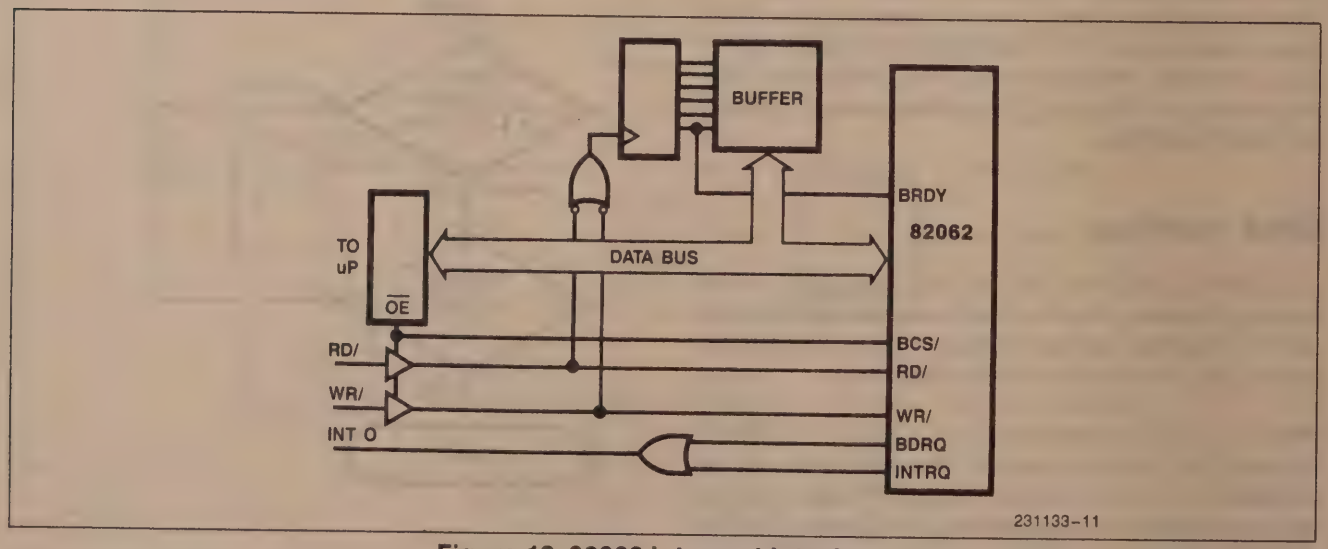


Figure 12. 82062 Interrupt Interface

Bit 2 - Aborted Command

This bit is set if a command was issued or in progress while DRDY (Pin 28) was deasserted or WR FAULT (Pin 30) was asserted. The Aborted Command bit will also be set if an undefined command is written into the COMMAND register, but an implied seek will be executed.

Bit 1 - TRACK 000

This bit is set only by the RESTORE command. It indicates that TRACK 000 (Pin 31) has not gone active after the issuance of 1024 stepping pulses.

Bit 0 - Data Address Mark

This bit is set during a READ SECTOR command if the Data Address Mark is not found after the proper Sector ID is read.

Reduce Write Current Register

This register is used to define the cylinder number where RWC (Pin 33) is asserted:

7	6	5	4	3	2	1	0
CYLINDER NUMBER / 4							

The value (0–255) written into this register is internally multiplied by 4 to specify the actual cylinder where RWC is asserted. Thus a value of 01H will cause RWC to activate on cylinder 4, 02H on cylinder 8 and so on. RWC will be asserted when the present cylinder is greater than or equal to the cylinder indicated by this register. For example, one ST506 compatible drive requires precompensation on cylinder 128 (80H) and above. Therefore the REDUCE WRITE CURRENT register should be loaded with 32 (20H). A value of FFH will keep the RWC output inactive regardless of the actual cylinder number.

Sector Count Register

This register is used to define the number of sectors that need to be transferred to the buffer during a READ MULTIPLE SECTOR or WRITE MULTIPLE SECTOR command.

7	6	5	4	3	2	1	0
# OF SECTORS							

The value contained in the register is decremented after each sector is transferred to/from the sector buffer. A zero represents a 256 sector transfer, a one a 1 sector transfer, etc. This register is ignored when single sector commands are specified in the Command register.

Sector Number

This register holds the sector number of the desired sector:

7	6	5	4	3	2	1	0
SECTOR NUMBER							

For a multiple sector command it specifies the first sector to be transferred. It is decremented after each sector is transferred to/from the sector buffer. The SECTOR NUMBER register may contain any value from 0 to 255. The ID Not Found bit will be set if the desired sector cannot be located on the track.

The SECTOR NUMBER register is also used to program the Gap 1 and Gap 3 lengths to be used when formatting a disk. See the WRITE FORMAT command description for further explanation.

Cylinder Number Low Register

This register holds the lower byte of the desired cylinder number:

7	6	5	4	3	2	1	0
LS BYTE OF CYLINDER NUMBER							

It is used in conjunction with the CYLINDER NUMBER HIGH register to specify a range of 0 to 1024 tracks.

Cylinder Number High Register

This register holds the two most significant bits of the desired cylinder number:

7	6	5	4	3	2	1	0
x	x	x	x	x	x	(9)	(8)

x = ignored

The 82062 contains a pair of registers that store the actual position where the R/W head are located. The CYLINDER NUMBER HIGH and LOW registers are considered the cylinder destination registers for seeks and other commands. The 82062 compares its internal registers to the destination registers and issues the number of steps in the right direction to make both sets of registers equal. After a command is executed, the internal cylinder position registers' contents are equal to the cylinder high/low registers. If a drive number change is detected on a new command, the 82062 automatically reads an ID field to update its internal cylinder position registers. This affects all commands except a RESTORE.

Pin Descriptions (continued)

Symbol	Pin. No.	Type	Name and Function
BDRQ	36	O	**Buffer Data Request:** Activated during Read or Write commands when a data transfer between the host and the 82062's sector buffer is required. Typically used as a DMA request line, or to generate an interrupt.
RD DATA	37	I	**Read Data:** Single ended input that accepts MFM data from the drive. See note 2.
RD GATE	38	O	**Read Gate:** Output that is high for data and ID fields. Goes active when DRUN has been high for 16 WR CLOCK periods to permit the external phase lock loop to lock onto the incoming disk data stream.
RD CLOCK	39	I	**Read Clock:** Clock input derived from the external data recovery circuits. See note 2.
V_{CC}	40	I	**D.C. Power:** +5V

NOTES:
1. This pin requires a pull-up resistor to function properly. A value of 1000 ohms will work satisfactorily.
2. This pin requires input levels that are not TTL compatible. These lines can be interfaced to TTL with a pull-up resistor. Too small of a resistor will produce a V_{IL} level that is too high. Too large of a resistor will degrade the signal's rise time. A minimum value for the resistor is determined as follows:

$$\frac{(V_{CC}\ \max) - (82062\ V_{IL}\ \max)}{(\text{TTL}\ I_{OL}\ \min.) - (82062\ I_{IL}\ \max)} = \text{Resistor}$$

TASK REGISTER FILE

The Task Register File is a bank of registers used to hold parameter information pertaining to each command. These registers and their addresses are:

A2	A1	A0	READ	WRITE
0	0	0	(Bus Tri-Stated)	(Bus Tri-Stated)
0	0	1	Error Flags	Reduce Write Current
0	1	0	Sector Count	Sector Count
0	1	1	Sector Number	Sector Number
1	0	0	Cylinder Low	Cylinder Low
1	0	1	Cylinder High	Cylinder High
1	1	0	SDH	SDH
1	1	1	Status Register	Command Register

NOTE:
Registers are not cleared by $\overline{\text{RESET}}$

Error Register

This read-only register contains specific error status after the completion of a command. If any bit in this register is set, then the Error bit in the Status Register will also be set. The bits are defined as follows:

7	6	5	4	3	2	1	0
BBD	CRC	–	ID	–	AC	TK000	DM

Bit 7 - Bad Block Detect

This bit is set when an ID field has been encountered that contains a bad block mark. The bad block bit is set only during formatting. The 82062 will terminate a command if an attempt is made to read a sector that contains this bit.

Bit 6 - CRC Data Field

This bit is set when a data field CRC error has occurred. The sector buffer may still be read but will contain errors.

Bit 5 - Reserved.

Not used. Set to zero.

Bit 4 - ID Not Found

This bit is set when the desired cylinder, head, sector or size parameter cannot be found after 8 revolutions of the disk, or if an ID field CRC error has occurred.

Bit 3 - Reserved.

Not used. Set to zero.

Sector/Drive/Head Register

The SDH register contains the desired sector size, drive number, and head number parameters. The format is shown below.

7	6 5	4 3	2 1 0
EXT	SECT SIZE	DRIVE	HEAD #

Both head number and sector size are compared against the disk's ID field. Head select and drive select lines are not available as outputs from the 82062 and must be generated externally.

Bit 7, the extension bit (EXT), is used to extend the data field by seven bytes when using ECC codes for READ/WRITE SECTOR commands. When EXT = 1, the CRC is not appended to the end of the data field and the data field becomes "sector size + 7" bytes long. The CRC is checked on the ID field regardless of the state of EXT. The SDH byte written into the ID field is different than the SDH Register contents. The reccorded SDH byte does not have the drive number (DRIVE) written but does have the BAD BLOCK mark written. The EXT bit must not be set during the Format command.

Note that use of the extension bit requires the gap lengths to be modified as described in the WRITE FORMAT command description.

Status Register

The status register is a read-only register which informs the host of certain events. This register is a flow-through latch until the microprocessor reads it at which point the drive status lines are latched. The INTRQ line will be reset when this register is read. The format is:

7	6	5	4	3	2	1	0
BUSY	READY	WF	SC	DRQ	–	CIP	ERROR

Bit 7 - Busy

This bit is set whenever the 82062 is transferring data between its sector buffer and the disk and reflects the state of the $\overline{BCS}$ pin. When $\overline{BCS}$ is active, the host should not access the sector buffer or any 82062 register. The 82062 will be generating a $\overline{RD}$ or $\overline{WR}$ pulse every 1.6 μsec and the host must not interfere with these data transfers. Busy is cleared when the data transfer operation is completed.

During other non-data transfer commands, Busy should be ignored as it will go active for short periods.

Bit 6 - Ready

This bit reflects the state of the DRDY (Pin 28) line at the time the microprocessor reads the status register. Transitions on the DRDY line will abort a command and set the aborted command bit in the error register.

Bit 5 - Write Fault

This bit reflects the state of the WR FAULT (Pin 30) line. Transitions on this line will abort a command and set the aborted command bit in the error register.

Short transitions on DRDY and WR FAULT may not show up in the status register. These pins are not latched until the microprocessor reads the status and by that time the error condition may have disappeared. However the aborted command bit will be set to notify the host of an error. To hold short transitions on these pins it is recommended that they be latched.

Bit 4 - Seek Complete

This bit reflects the state of the SC (Pin 32) line. Commands which initiate a seek will pause until Seek Complete is set.

Bit 3 - Data Request

The Data request bit (DRQ) reflects the state of the BDRQ (Pin 36) line. It is set when the sector buffer should be loaded with data or read by the host processor, depending upon the command. The DRQ bit and the BDRQ line remain high until BRDY is sampled, indicating the operation has completed.

Bit 2 - Reserved

Not used. Set to zero.

Bit 1 - Command in Progress

When this bit is set, a command is being executed and a new command should not be loaded until it is cleared. Although a command may be executing, the sector buffer is still available for access by the host processor. If CIP is set, only the status register can be read regardless of which register is selected.

Bit 0 - Error

This bit is an OR of the contents of the error register. Any bit being set in the error register sets this bit. This bit is cleared when a new command is loaded.

Command Register

This write-only register is loaded with the desired command:

7	6	5	4	3	2	1	0
COMMAND							

The 82062 begins to execute immediately upon loading any value into this register. This register should not be written while the Busy or Command in Progress bits are set in the STATUS register. The INTRQ line (Pin 3) if set, will be cleared by a write to the COMMAND register.

Instruction Set

The 82062 WDC instruction set contains six commands. Prior to loading the command register, the host processor must first set up the Task Register File with the information needed for the command. Except for the COMMAND register, the registers may be loaded in any order. If a command is in progress, a subsequent write to the COMMAND register will be ignored. A command is finished when the command in progress (CIP) bit in the STATUS register is cleared. See the Command Section for an explanation of each command.

COMMAND	7	6	5	4	3	2	1	
RESTORE	0	0	0	1	R3	R2	R1	R0
SEEK	0	1	1	1	R3	R2	R1	R0
READ SECTOR	0	0	1	0	I	M	0	T
WRITE SECTOR	0	0	1	1	0	M	0	T
SCAN ID	0	1	0	0	0	0	0	T
WRITE FORMAT	0	1	0	1	0	0	0	0

R 3−0 = Rate Field

For 5 MHz WR Clock:

0000 — ≈35 μs
0001 — 0.5 ms
0010 — 1.0 ms
0011 — 1.5 ms
0100 — 2.0 ms
0101 — 2.5 ms
0110 — 3.0 ms
0111 — 3.5 ms
1000 — 4.0 ms
1001 — 4.5 ms
1010 — 5.0 ms
1011 — 5.5 ms
1100 — 6.0 ms
1101 — 6.5 ms
1110 — 7.0 ms
1111 — 7.5 ms

COMMAND	7	6	5	4	3	2	1
T = Retry Enable							
T = 0 Enable Retries T = 1 Disable Retries							
M = Multiple Sector Flag							
M = 0 Transfer 1 Sector M = 1 Transfer Multiple Sectors							
I = Interrupt Enable							
I = 0 Interrupt at BDRQ time I = 1 Interrupt at end of command							

Programming the 82062

This section consists of two parts. The first part gives an explanation of each command, a flowchart showing the 82062's sequence of events, and the commands' sequence of events as seen by the host microprocessor. The second section shows flowcharts of general software routines and their PLM equivalent, for both polled and interrupt driven software.

The designer must remember that the 82062 expects a full sector buffer that can be isolated from the host during data transfers between the 82062 and the disk. Since the 82062 assumes a full sector buffer is available, it does not check for data overrun or underrun error conditions. If such a condition occurs, corruption of data will happen and the host will have no indication of an error. The design must guarantee against over-run and under-run conditions when not using the sector buffer approach.

Commands

A command is placed into the command register only after the Task Registers have been written with proper values. The Task Registers may be loaded in any order. A command, once started, can only be terminated by a hardware reset to the 82062. This may corrupt data on the disk by removing necessary control signals out of sequence.

The general sequence of a command is as follows:

— The host loads the Task Registers
— The host loads the Command Register
— The 82062 locates the correct cylinder
— Data transfer takes place
— The 82062 issues an interrupt

Restore Command – 0 0 0 1 R3 R2 R1 R0

The Restore command is used to position the heads to cylinder 0. This command must be issued to the 82062 on power-up to initialize internal registers. The user specified rate field (R3–R0) is stored internally for FUTURE use in commands with implied seeks. The step rate value is not used with this command. The actual stepping rate used is dependent upon the handshake delay between the 82062 issuing a step pulse and the drive returning a seek complete for each track (roughly 20 ms). After each step pulse is issued, the 82062 waits for a rising edge on the Seek Complete (SC) line before issuing the next pulse. If 8 index pulses are received without a rising edge on SC, the 82062 will switch to sampling the level of the SC line. If after 1,024 step pulses the Track 00 signal has not gone active, the 82062 will terminate the command and set the TRACK 000 bit in the Error Register. The command will terminate if WR Fault goes active or DRDY goes inactive at any time. Figure 13 is a flow chart of the command.

This command should precede the format command. The format command will be aborted if an ID field is not present (because the disk was never formatted) and

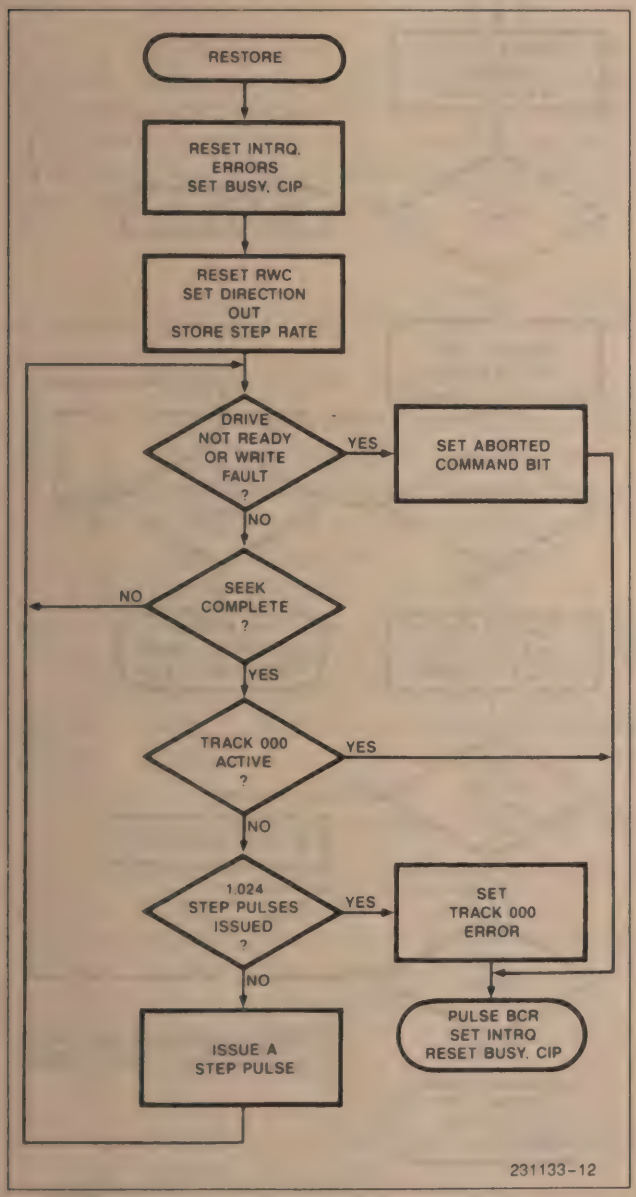


Figure 13. Restore Command Flow

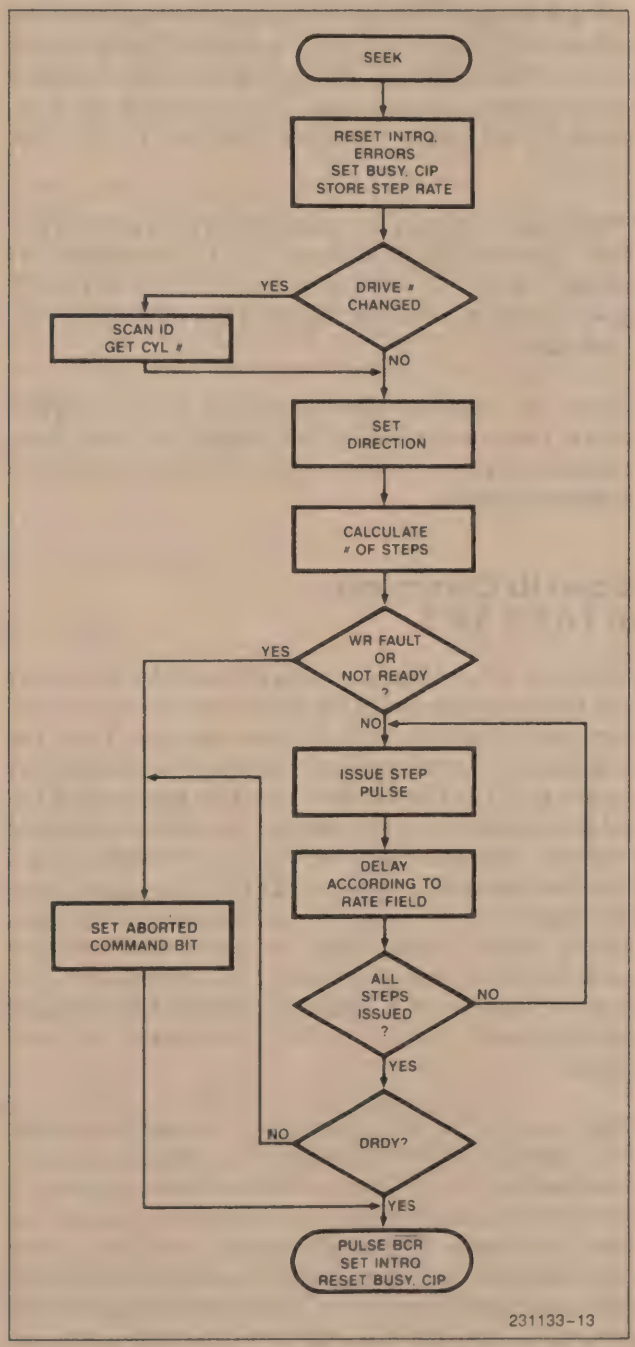


Figure 14. Seek Command Flow

a new drive is selected. Recall the 82062 will do a Scan ID to update internal registers when the drive is changed. This information is used to calculate the number of steps required to get to the destination cylinder. When the heads are positioned to track zero the 82062 will not try to read an ID field, but will issue the correct number of steps.

Seek Command – 0 1 1 1 R3 R2 R1 R0

The Seek command positions the heads to the cylinder specified in the Task Registers. The direction and number of step pulses issued is calculated by comparing the cylinder high/low registers to an internal "present position" cylinder register. The present position register is updated after all step pulses are issued and the command is terminated. The Seek Complete input is not checked.

The actual stepping rate is taken from the rate field bits (R3–R0) and stored for future use. The command terminates at once if WR FAULT goes active or DRDY goes inactive at any time. Figure 14 is a flowchart of the command.

Since the data transfer commands feature implied seeks, this command is of use mainly to those using multiple drives and software that can take advantage of overlapped seeks.

Scan ID Command – 0 1 0 0 0 0 0 T

The Scan ID command is used by both the 82062 and the host to update the SDH, the Sector Number, Cylinder and internal present position registers. Once the command is issued, the Seek Complete line is sampled until valid. The first ID field found, as indicated by the address mark, is loaded into the previously mentioned registers. The Bad Block bit will be set if detected, and the command will terminate. ID CRC errors will start the search sequence over for a maximum of 10 index pulses, but the registers will be loaded with whatever data the 82062 had perceived as ID information. Improper states on WR Fault on DRDY will terminate the command. Figure 15 is the flow chart of the command.

The main use for this command is to determine where the heads are currently located and what size the sectors are (i.e. 256, 512 etc.). Without this command, it would be necessary to recall the heads to track zero and then step out to the desired cylinder each time a drive was changed. Specifying the wrong sector size would yield an ID not found error. This command enables the system to read the disk drive to determine what size sectors were recorded.

Read Sector Command – 0 0 1 0 I M 0 T

The READ SECTOR command is used to transfer one or more sectors of data from the disk to the sector buffer. Upon receipt of the READ SECTOR command, the 82062 checks the CYLINDER NUMBER LOW/HIGH register pair against an internal cylinder position register to see if they are equal. If not, the direction and number of steps are calculated and a seek takes place. If an implied seek is performed, the 82062

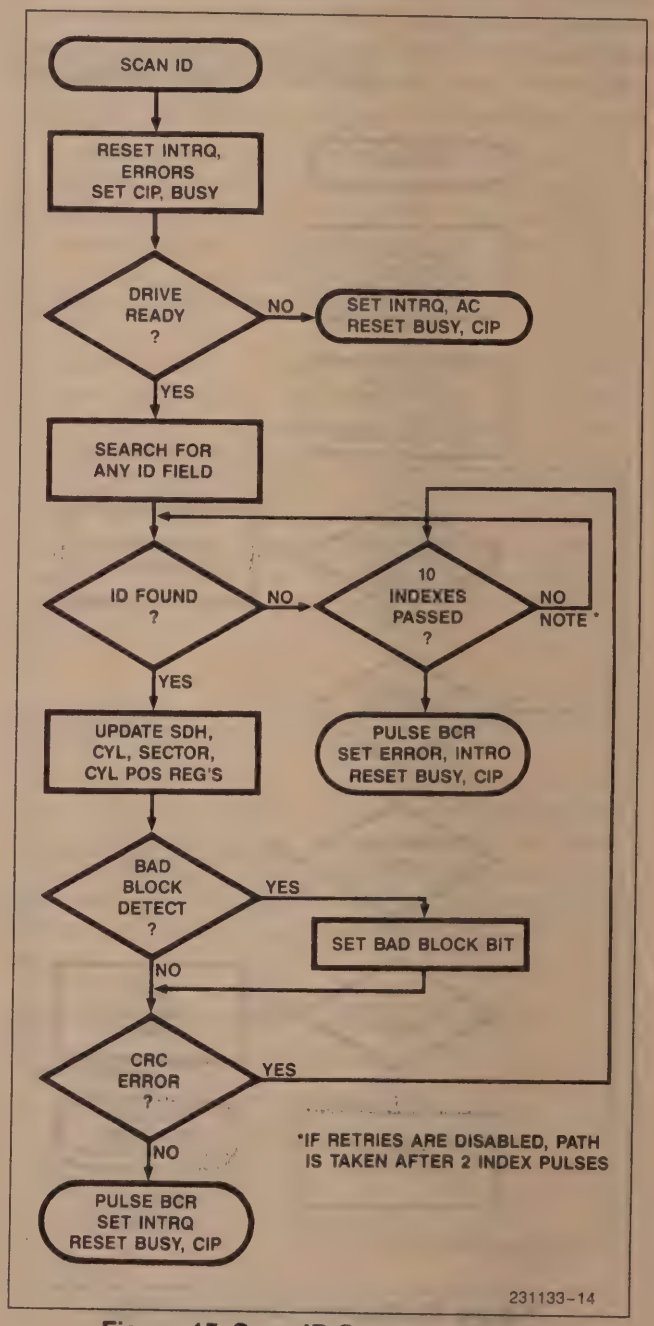


Figure 15. Scan ID Command Flow

will search until a rising edge of SC is received. The WR FAULT and DRDY lines are monitored throughout the command.

Once the Seek Complete (SC) line is high (with or without an implied seek having occurred), the search for an ID field begins. If T = 0 (retries enabled), the 82062 must find an ID with the correct cylinder number, head, sector size, and CRC within 10 revolutions, or a Scan ID and re-Seek will be performed. The search for the proper ID will again be tried for up to 10 revolutions. If the correct sector is still not found, the appropriate error bits will be set and the command terminated. Data CRC errors will also be retried for up to 10 revolutions (if T = 0).

If T = 1 (retries disabled), the ID search must find the correct sector within 2 revolutions or the appropriate error bits will be set and the command terminated.

Both the READ SECTOR and WRITE SECTOR commands feature a "simulated completion" to ease programming. DRQ/BDRQ will be generated upon detecting an error condition. This allows the same program flow for successful or unsuccessful completion of a command.

When the data address mark is found, the 82062 is ready to tranfer data to the sector buffer. After the data has been transferred, the I bit is checked. If I = 0, INTRQ is made active coincident with BDRQ, indicating that a transfer of data from the buffer to the host processor is required. If I = 1, INTRQ will occur at the end of the command, i.e. after the buffer is unloaded by the host.

The M bit is set for multiple sector transfers. When M = 0, one sector is transferred and the SECTOR COUNT register is ignored. When M = 1, multiple sectors are transferred. After each sector is transferred, the 82062 decrements the SECTOR COUNT register and increments the SECTOR NUMBER register. The next logical sector will be transferred regardless of any interleave. Sectors are numbered at format time.

Multiple sector transfers continue until the SECTOR COUNT register equals zero, or the BRDY line goes active (low to high). If the SECTOR COUNT register is non-zero (indicating more sectors are to be transferred but the buffer is full), BDRQ will be made active and the host must unload the buffer. After this occurs, the buffer will again be free to accept the remaining sectors from the 82062. This scheme enables the user to transfer more sectors than the buffer memory has capacity for.

In summary then, READ SECTOR operation is as follows:

When M = 0 (READ SECTOR)

(1)	Host:	Sets up parameters; issues READ SECTOR command.
(2)	82062:	Strobes $\overline{BCR}$; sets $\overline{BCS}$ = 0.
(3)	82062:	Finds sector specified; transfers data to buffer.
(4)	82062:	Strobes $\overline{BCR}$; sets $\overline{BCS}$ = 1.
(5)	82062:	Sets BDRQ = 1; DRQ = 1.
(6)	82062:	If I bit = 1 go to (9).
(7)	Host:	Reads contents of sector buffer.
(8)	82062:	Waits for BRDY, then sets INTRQ = 1: END.
(9)	82062:	Sets INTRQ = 1.
(10)	Host	Reads out contents of buffer; END.
(11)	82062:	If I = 1 wait for BRDY, then clear BDRQ; END.

When M = 1 (READ MULTIPLE SECTOR)

(1)	Host:	Sets up parameters; issues READ SECTOR command.
(2)	82062:	Strobes $\overline{BCR}$; sets $\overline{BCS}$ = 0.
(3)	82062:	Finds sector specified; transfers data to buffer.
(4)	82062:	Decrements SECTOR COUNT register; increments SECTOR NUMBER register.
(5)	82062:	Strobes $\overline{BCR}$; sets $\overline{BCS}$ = 0.
(6)	82062:	Sets BDRQ = 1; DRQ = 1.
(7)	Host:	Reads out contents of buffer.
(8)	82062:	Waits for BRDY.
(9)	82062:	When BRDY = 1, if Sector Count = 0 then go to (11).
(10)	82062:	Go to (2).
(11)	82062:	Set INTRQ = 1; End.

A flowchart of the READ SECTOR command is shown in Figures 16A and 16B.

Write Sector Command – 0 1 1 1 0 M 0 T

The WRITE SECTOR command is used to write one or more sectors of data to the disk from the sector buffer. Upon receipt of a WRITE SECTOR command the 82062 checks the CYLINDER NUMBER LOW/HIGH register pair against the internal cylinder position register to see if they are equal. If not, the direction and number of steps calculation is performed and a seek takes place. The WR FAULT and DRDY lines are checked throughout the command.

When the Seek Complete (SC) line is found to be true (with or without an implied seek having occurred), the BDRQ signal is made active and the host proceeds to load the buffer. Once BRDY goes high, the ID field with the specified cylinder number, head, and sector size is searched for. Once found, WR GATE is made active and the data is written to the disk. If retries are enabled (T = 0), and if the ID field cannot be found within 10 revolutions, a Scan ID and re-Seek are performed. If the correct ID field is not found within 10 additional revolutions, the ID Not Found error bit is set and the command is terminated. If retries are disabled, (T = 1) and if the ID field cannot be found within 2 revolutions, the ID Not Found error bit is set and the command is terminated.

During a WRITE MULTIPLE SECTOR command (M = 1), the SECTOR NUMBER register is decremented and the SECTOR COUNT register is incremented after the transfer to the disk takes place. During multiple sector transfers if BRDY is asserted after the first sector is transferred from the buffer, the 82062 will transfer the next sector before issuing BDRQ. The 82062 will set BDRQ and wait for the host processor to place more data in the buffer.

In summary then, the WRITE SECTOR operation is as follows:

When M = 0, 1 (WRITE SECTOR)

(1)	Host:	Sets up parameters; issues WRITE SECTOR command.
(2)	82062:	Sets BDRQ = 1, DRQ = 1.
(3)	Host:	Loads sector buffer with data.
(4)	82062:	Waits for BRDY = 0 to 1.
(5)	82062:	Finds specified ID field; writes sector to disk.
(6)	82062:	If M = 0, then set INTRQ = 1; END.
(7)	82062:	Increment SECTOR NUMBER register; decrement SECTOR COUNT register.
(8)	82062:	If SECTOR = 0, then set INTRQ = 1; END.
(9)	82062:	Go to (2).

A flowchart of the WRITE SECTOR command is shown in Figure 17.

Write Format Command 0 1 0 1 0 0 0 0

The WRITE FORMAT command is used to format one track using the Task Register File and the sector buffer. During execution of this command, the sector buffer is used for additional parameter information instead of sector data. Shown in Figure 18 is the contents of the sector buffer for a 32 sector/track format with an interleave factor of two. Each sector requires a two byte sequence. The first byte designates whether a bad block mark is to be recorded in the sector's ID field. A 00 Hex is normal; an 80H indicates a bad block mark for the sector. In the example of Figure 18, sector 04 will get a back block mark recorded. Any attempt to access sector 4 in the future will terminate the command.

The second byte indicates the logical sector number to be recorded. This allows sectors to be recorded with any interleave factor desired. The remaining memory in the sector buffer may contain any value. Its only purpose is to generate a BRDY to tell the 82062 to begin formatting the track. An implied seek is in effect on this command. As for other commands, if the drive number has been changed an ID field will be scanned for cylinder position information before the implied seek is performed. If no ID field can be read (because the track had been erased or because an incomplete format had been used), an ID Not Found error will result and the WRITE FORMAT command will be aborted. This can be avoided by issuing a RESTORE command before formatting.

The SECTOR COUNT register is used to hold the total number of sectors to be formatted (01H = 1 sector; 00H = 256 sectors), while the SECTOR NUMBER register holds the number of bytes (minus three) to be used for Gap 1 and Gap 3. For instance, if the SECTOR COUNT register value is 02H and the SECTOR NUMBER register value is 00H, then 2 sectors are written on a track and 3 bytes of 4EH are written for Gap 1 and Gap 3. The data fields are filled with FFH and the CRC is automatically generated and appended. All gaps are filled with 4EH. After the last sector is written, the track is filled with 4EH until the index pulse terminates the write. The Gap 3 value is deter-

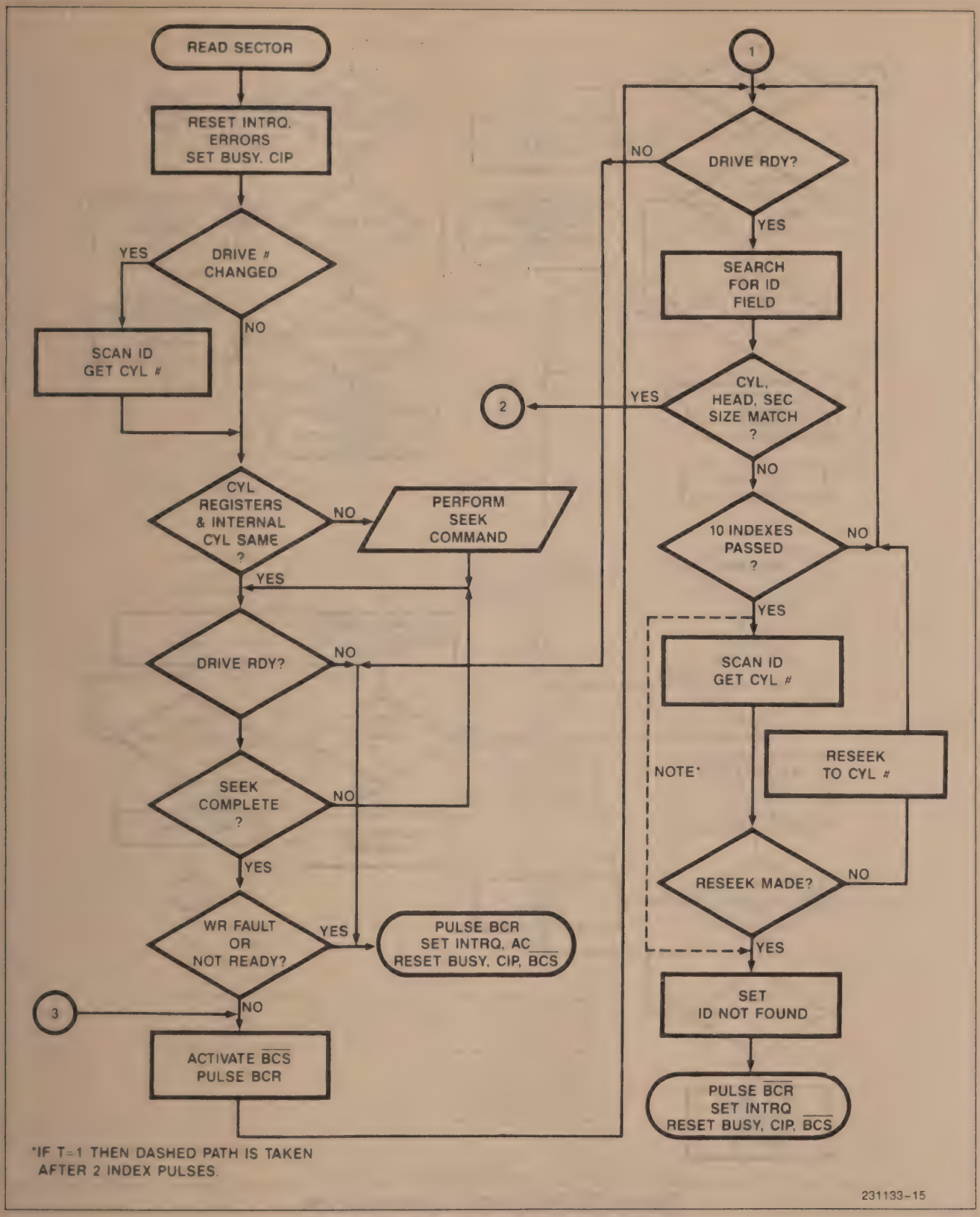


Figure 16A. Read Sector Command Flow

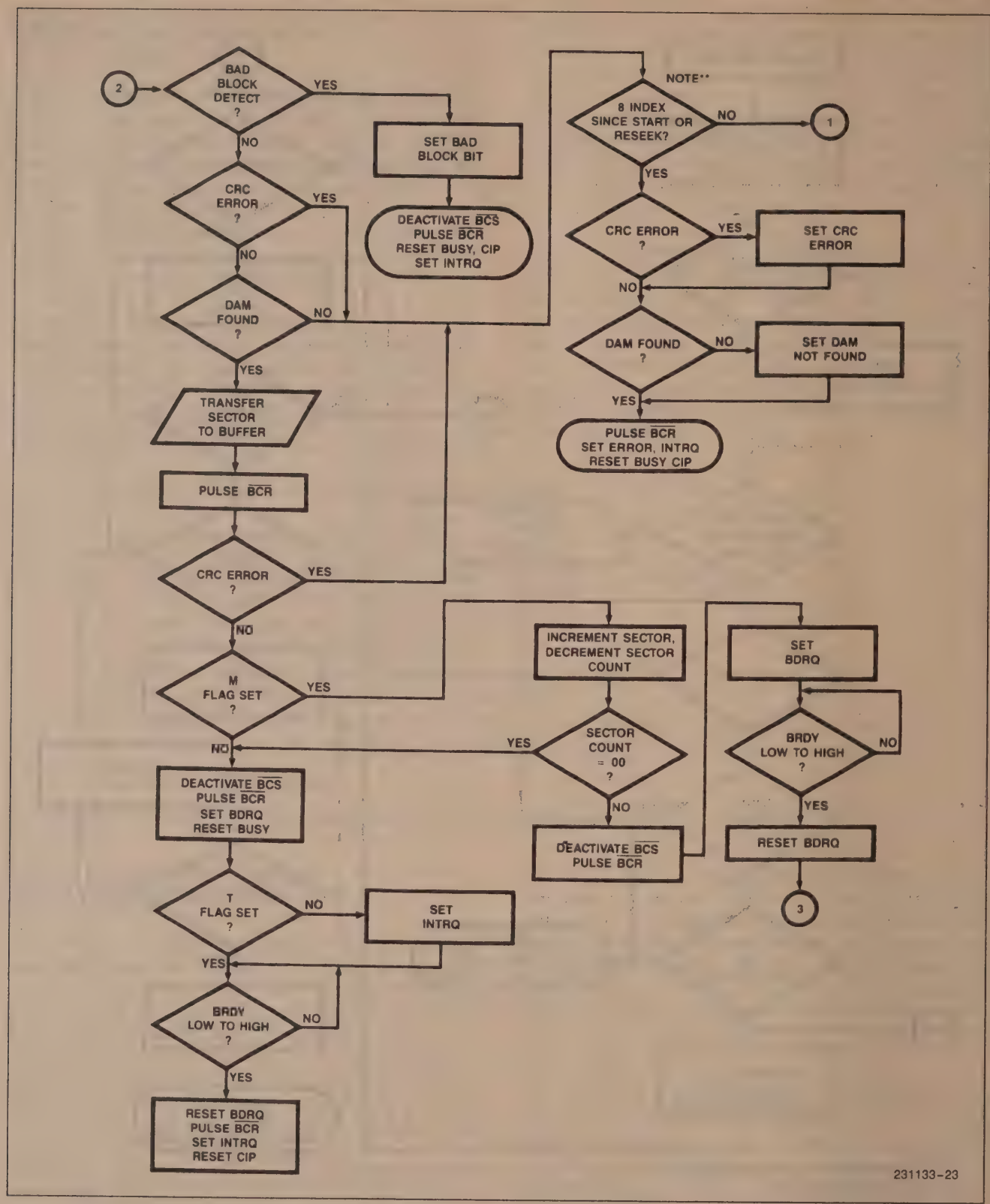


Figure 16B. Read Sector Command Flow

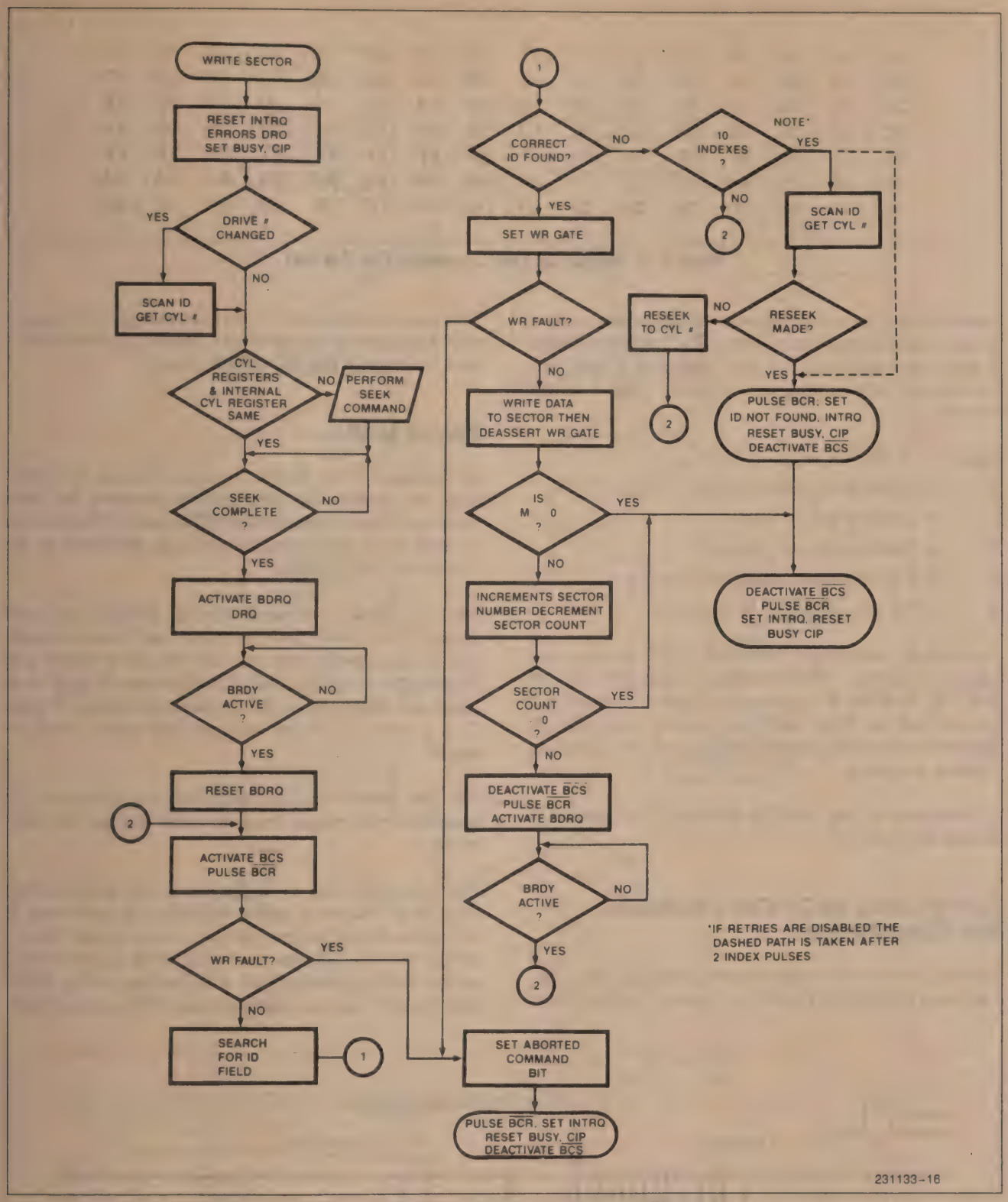


Figure 17. Write Sector Command Flow

00	00	00	10	00	01	00	11	00	02	00	12	00	03	00	13
80	04	00	14	00	05	00	15	00	06	00	15	00	07	00	17
00	08	00	18	00	09	00	19	00	0A	00	19	00	0B	00	1B
00	0C	00	1C	00	0D	00	1D	00	0E	00	1E	00	0F	00	1F
FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Figure 18. Sector Buffer Contents For Format

mined by the drive motor speed variation, data sector length, and the interleave factor. The interleave factor is only important when 1:1 (no) interleave is used. The formula for determining the minimum Gap 3 length value is:

Gap 3 = (2 * M * S) + K + E

M = motor speed variation (e.g., 0.03 for ±3%)

S = sector length in bytes

K = 25 for interleave factor of 1

K = 0 for any other interleave factor

E = 7 if the sector is to be extended

As with all commands, a WR FAULT or drive not ready condition, will terminate execution of the WRITE FORMAT command. Figure 19 shows the format that the 82062 will write on the disk. The extend bit in the SDH register must not be set during the Format command.

A flowchart of the WRITE FORMAT command is shown in Figure 20.

SOFTWARE SECTION: GENERAL PROGRAMMING

This section describes the software needed to communicate with the 82062 in order to store and retrieve data. This chapter describes the software in a general manner and Appendix B contains the actual implementation used to exercise the 82062 SBX board.

Polled Mode

As discussed in the Polled Interface Section, the 82062 does not directly support polled operation for data transfers without the addition of hardware. This section is based upon the polled interface as described in the Polled Interface Section.

The six 82062 commands can be divided into two groups, those with data transfers and those without. The commands that do not use the sector buffer are: Restore, Seek and Scan ID. The functions of each command are explained in the Commands Section. Figure 21 is a flowchart of a polled operation and a PLM example.

The last status that was read will contain any error conditions that might have occurred during the command.

For commands that do make use of the sector buffer, the size of the sector buffer will affect the software. If the sector buffer is equal in size to one sector, then a carry out of an address counter (for the sector buffer) as the buffer is being filled will indicate to the 82062 that the command should continue. If the sector buffer

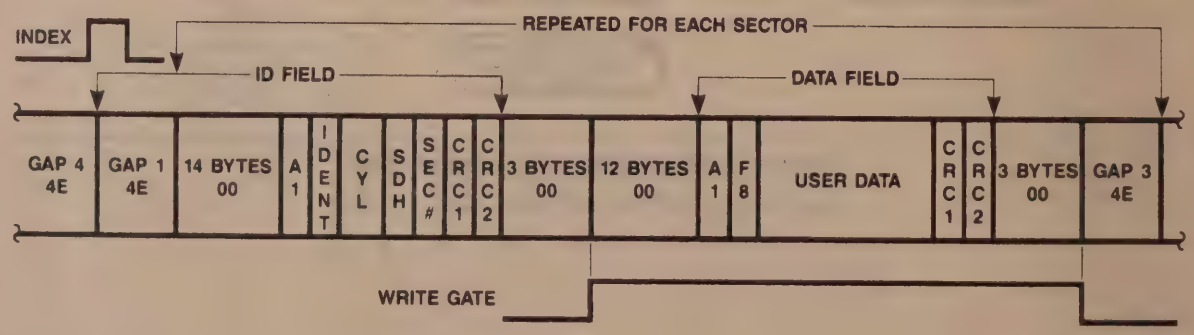


Figure 19. 82062 Sector Format

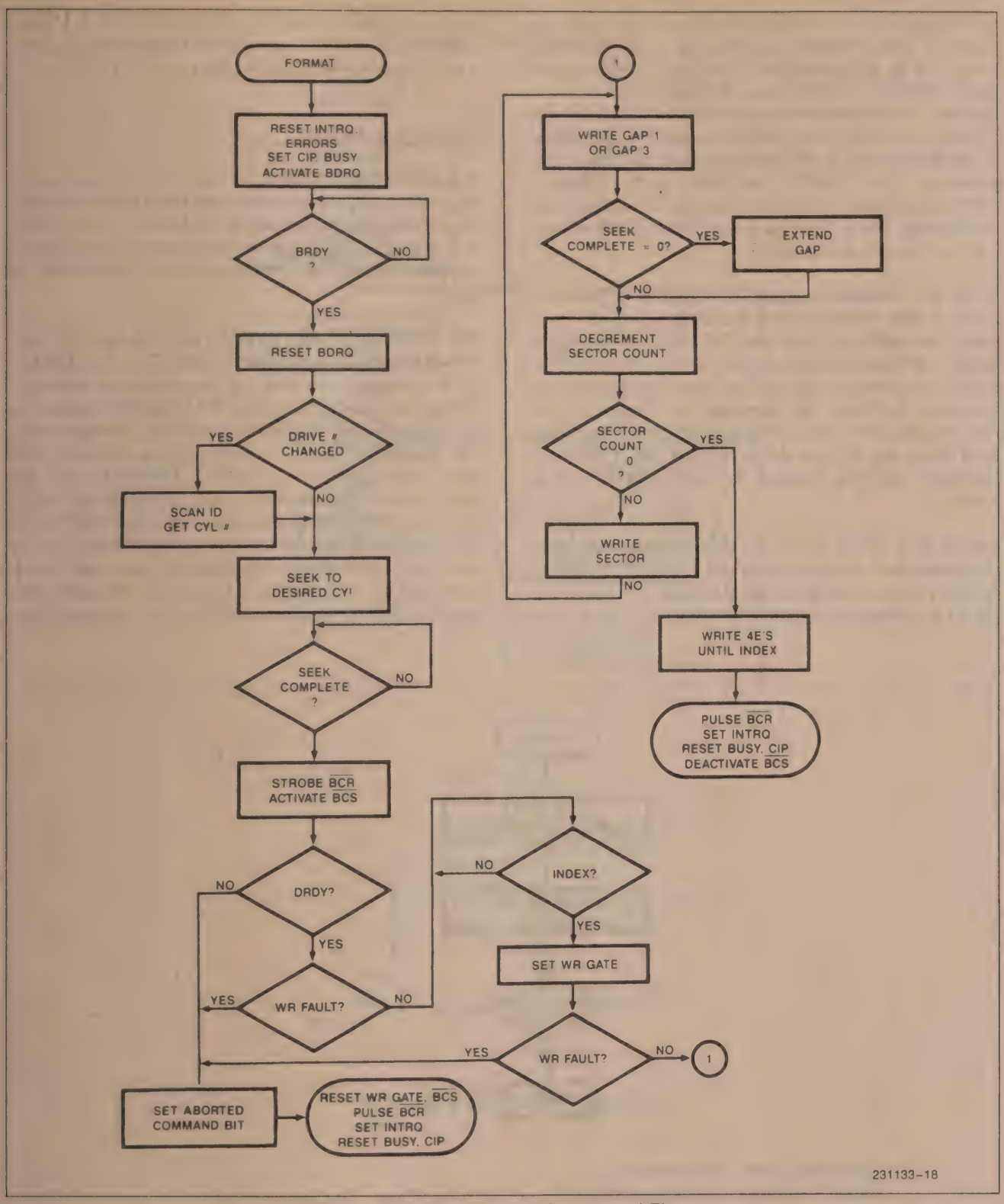


Figure 20. Write Format Command Flow

size is equal to two or more disk sectors, and only one sector is being transferred, then the carry out signal would not go active, and the 82062 will be forever waiting for BRDY. In this case an I/O port would have to be used to generate this signal for the 82062 so that command execution can finish. Figure 22 is a flowchart of the READ SECTOR command, and its PLM representation. The WRITE SECTOR and FORMAT TRACK commands are equivalent in terms of software interfacing. Their flowcharts and their PLM equivalents are shown in Figure 23.

Once the command register is written the 82062 requests a data transfer before locating the proper track. Once the buffer is filled and BRDY is asserted, the 82062 will locate the target track and sector. If the ID is not located before the selected number of retries have occurred, the 82062 will terminate the command. The data transferred to the sector buffer will not have been used. Once the command has finished (i.e., CIP = 0), the status and error registers will inform the host of an error.

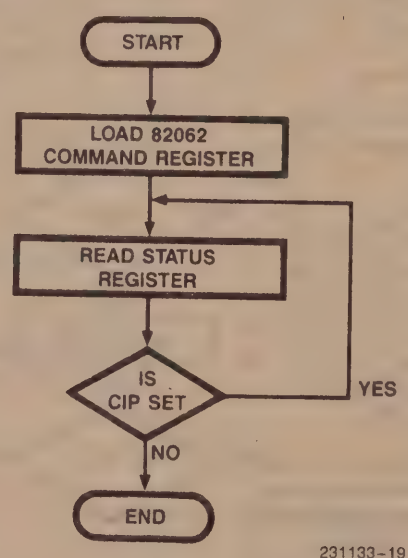
Figure 24 is the PLM routine that allows for all six of the commands. It differs from the READ and WRITE routines in that the direction that data is to be transferred is determined by the command.

Figure 24 also works for multiple sector transfers. However, the BRDY signal must be generated in hardware (the carry-out of an address counter).

Interrupt Mode

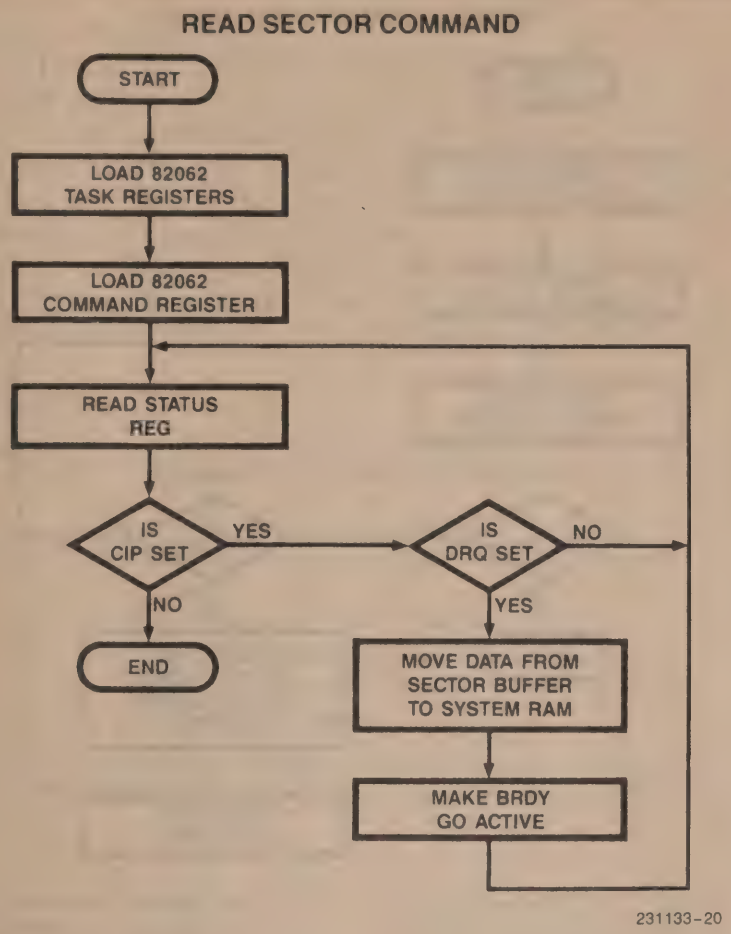
Interrupt driven software is chosen when the microprocessor must execute other tasks and cannot sit waiting for the disk to reposition its heads, as in a polled environment. The delay in repositioning heads can be anything from a couple of milliseconds to a second or more.

The 82062's interrupt (INTRQ) pin goes active to indicate that the command has finished. The READ SECTOR command provides the programmable choice of having the interrupt occur at the end of the data transfer or the normal end of the command. The reason for this option is that when the 82062 signals that a data transfer is required (via BDRQ, DRQ) the disk has been read and the data has been placed in the buffer. The host would remove the data and issue BRDY. The 82062 would then issue an interrupt indicating that the command has finished. The interrupt procedure would only have to read the status register. If the interrupt is issued at BDRQ the host would remove the buffer data




```
Disk$Operation: Procedure;
   Call Write$82062$Task$Reg's; /* Write Task Registers */
   Output (Command$Reg) = Command;
   Status = Input (Status$Reg); /* Read Status Reg */
   Do while Status and CIP = CIP; /* Wait until command finishes */
      Status = Input (Status$Reg);
   End;
End Disk$Operation;
```

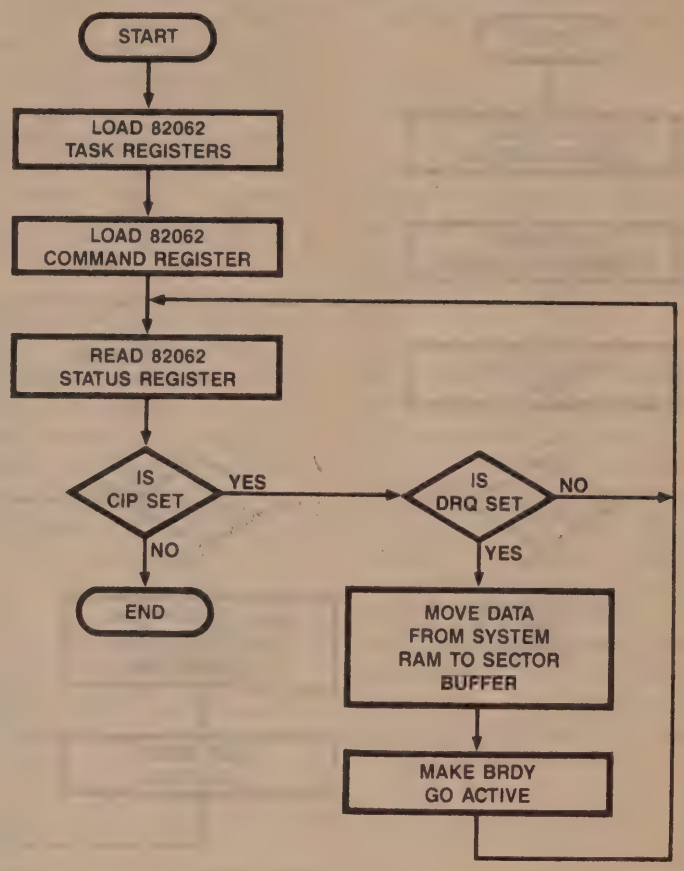
Figure 21. Polling Status




```
Disk$Operation: Procedure;
    Call Write$82062$Task$Regs;
    Output (Command $ Reg) = Command;
    Status = Input (Status$Reg);
    Do while Status and CIP = CIP;
        If Status and DRQ = DRQ then Do;
          Call Read$Data$From$Buffer;
          Output (BRDY$PORT) = 01;
        End;
        Status = Input (Status$Port)
    End;
End Disk$Operation;
```

Figure 22. Polling For Read Data

WRITE, FORMAT COMMANDS




```
Disk$Operation: Procedure;
  Call Write$82062$Task$Regs;
  Output (Command$Reg) = Command;
  Status = Input (Status$Reg);
  Do while status and CIP = CIP;
    If status and DRQ = DRW then do;
      Call Write$Data$to$Buffer;
      Output (BRDY$Port) = 01; /* Make BRDY go active */
    End;
    Status = Input (Status$Reg)
  End;
End Disk$Operation;
```

Figure 23. Polling For Write Data

```
Disk$Operation: Procedure;
   Call Write$82062$Task$Regs; /* Write registers */
   Output (Command$Reg) = Command; /* Start command */
   Status = Input (Status$Reg); /* Read status */
   Do while status and CIP = CIP; /* Is a command in progress */
     If status and DRQ = DRQ then do; /* Data transfer? = yes */
       If command = Read$Sector then
         Call Read$Data$From$Buffer; /* Remove data */
       Else Call Write$Data$to$Buffer; /* Send data */
       Output (BRDY$PORT) = 01; /* Toggle BRDY 0 to 1 */
     End;
End Disk$Operation;
```

Figure 24. Complete Polled Flow

```
Start$Disk$Operation: Procedure;
    Call Write$82062$Task$Reg's;
    Output (Command $ Reg) = Command;
End Start$Disk$Operation;
```

Figure 25. Interrupt Mode; Starting a Disk Transfer

and generate BRDY. At this point the status and error registers contain valid information. Generating an interrupt at BDRQ time may save some systems some software overhead.

The WRITE SECTOR and FORMAT commands do not have this option because the sector buffer is filled before the track and sector are located. Hence, there can be significant delays between asking for data and the command terminating.

In an interrupt driven environment, the 82062 can interface to a DMA controller for data transfers between the sector buffer and the host's RAM. If a DMA controller is not available an interrupt must be generated via the BDRQ line. However, BDRQ can stay active for long periods of time (until BRDY is generated). The interrupt sensing logic must take this into account to avoid being retriggered constantly. Intel's 8259A Interrupt Controller 8259A provides that capability. It should be programmed for edge triggered interrupts or the end of interrupt byte must not be issued until BDRQ is removed to prevent retriggering.

Figure 25 is a PLM example of starting a disk operation in an interrupt driven environment. The command starts, and some indefinite amount of time later an interrupt would be generated, indicating service is required.

If a DMA controller is used, it would have to be programmed and initialized before the command is issued to the 82062. Recall that once a data transfer between the microprocessor and 82062 has finished, BRDY must be set high. As long as BRDY is generated from hardware, no microprocessor intervention is needed. If BRDY is generated by an I/O port the microprocessor will have to perform this function (this will be the case with any system that has a sector buffer larger than one sector). (One option could be to generate an interrupt from the terminal count pin of the DMA controller. The microprocessor would then issue a BRDY.) Data transfers between host RAM and the sector buffer would be handled without microprocessor intervention. The interrupt would then signal that the command has finished as shown in Figure 26. The only operation the host processor would perform is to check the status register of the 82062 for any error conditions.

If BDRQ is used to generate an interrupt in addition to the normal interrupt, then the routines shown in Figure 27 will check the status register to see if a data transfer should be executed or if the command is finished. If DRQ is not set, the command has finished and any error conditions would be in the status register.

Another possibility would be to have separate interrupt routines for the two possible sources of interrupts

```
End$of$Transfer: Procedure Interrupt;
    Status = Input (Status$Register);
    Output (8259A PIC) = End$of$Interrupt;
End End$of$Transfer;
```

Figure 26. Checking Status via Interrupt

```
Service$Disk$Controller: Procedure Interrupt;
    Status = Input (Status$Port);
    If Status and DRQ = DRQ then
        Call Transfer$Data$To/From$Buffer;/* Enable DMAC */
    Output (8259A PIC) = End$of$Interrupt;
End Service$Disk$Controller;
```

Figure 27. Complete Interrupt Procedure

(INTRQ, BRDQ). There would then be no need to test the status to see which interrupt had occurred.

APPLICATION EXAMPLE

This section shows an application using the 82062 interfaced to the SBX bus. A quick overview of the SBX bus is provided (pin descriptions, general wave forms) as a background for the application. Designing the 82062 onto an SBX Multimodule board was chosen to highlight the size and complexity differences between earlier TTL, MSI, LSI-based disk controller boards and what is possible using the 82062. Both the hardware and software sections will be applicable to most other designs using the 82062. This design example is called SBX82062 and does not represent a real product offered by Intel Corporation. Appendix C contains the schematic of the SBX board.

The advantage of the SBX Multimodule is that it permits the system to be tailored for specific needs with a minimum of effort. The advantage of an SBX based disk controller is that a current system can make use of the capacity, reliability and speed of a hard disk with no (or minimal) hardware redesign.

iSBX Bus Multimodule Boards

The iSBX Multimodule boards are small, specialized, I/O mapped boards which plug onto base boards. The iSBX boards connect to the iSBX bus connector and convert the iSBX bus signals to a defined I/O interface.

Base Boards

The base board decodes I/O addresses and generates the chip selects for the iSBX Multimodule boards. In 8-bit systems, the base board decodes all but the lower three addresses in generating the iSBX Multimodule board chip selects. In 16-bit systems, the base board decodes all but the lower order four addresses in generating the iSBX Multimodule board chip selects. Thus, a base board would normally reserve two blocks of 8 I/O ports for each iSBX socket it provides.

There are two classes of base boards, those with Direct Memory Access (DMA) support and those without. Base boards with DMA support are boards with DMA controllers on them. These boards, in conjunction with an iSBX Multimodule board (with DMA capability), can perform direct I/O to memory or memory to I/O operations.

iSBX Bus Interface

The iSBX bus interface can be grouped into six functional classes:

1. Control Lines
2. Address and Chip Select Lines
3. Data Lines
4. Interrupt Lines
5. Option Lines
6. Power Lines

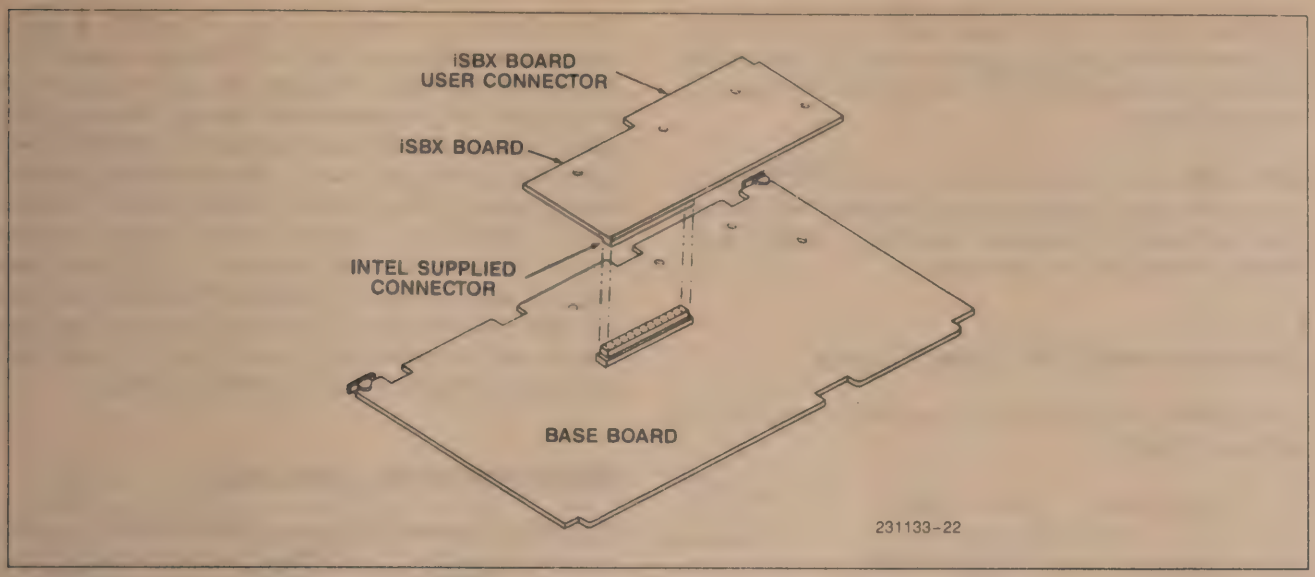


Figure 28. iSBX Multimodule Board Concept (Double Wide)

Control Lines

The following signals are classified as control lines:

COMMANDS:
- $\overline{\text{IORD}}$ (I/O Read)
- $\overline{\text{IOWRT}}$ (I/O Write)

DMA:
- MDRQT (DMA Request)
- $\overline{\text{MDACK}}$ (DMA Acknowledge)
- TDMA (Terminate DMA)

INITIALIZE:
- RESET

CLOCK:
- MCLK (iSBX Multimodule Clock)

SYSTEM CONTROL:
- $\overline{\text{MWAIT}}$
- $\overline{\text{MPST}}$ (iSBX Multimodule Board Present)

Command Lines ($\overline{\text{IORD}}$, $\overline{\text{IOWRT}}$)

The command lines are active low signals which provide the communication link between the base board and the iSBX Multimodule board. An active command line, conditioned by chip select, indicates to the iSBX Multimodule board that the address lines are valid and the iSBX Multimodule board should perform the specified operation.

DMA Lines (MDRQT, $\overline{\text{MDACK}}$, TDMA)

The DMA lines are the communication link between the DMA controller device on the base board and the iSBX Multimodule board. MDRQT is an active high output signal from the iSBX Multimodule board to the base board's DMA device requesting a DMA cycle. $\overline{\text{MDACK}}$ is an active low input signal to the iSBX Multimodule board from the base board DMA device acknowledging that the requested DMA cycle has been granted. TDMA is an active high output signal from the iSBX Multimodule board to the base board. TDMA is used by the iSBX Multimodule board to terminate DMA activity. The use of the DMA lines is optional as not all base boards will provide DMA channels and not all iSBX Multimodule boards will be capable of supporting a DMA channel.

Initialize Lines (Reset)

This input line to the iSBX Multimodule board is generated by the base board to put the iSBX Multimodule board into a known internal state.

Clock Lines (MCLK)

This input to the iSBX Multimodule board is a timing signal. The 10 MHz (+0%, −10%) frequency can vary from base board to base board. This clock is asynchronous from all other iSBX bus signals.

System Control Lines ($\overline{\text{MWAIT}}$, $\overline{\text{MPST}}$)

These output signals from the iSBX Multimodule board control the state of the system.

An active $\overline{\text{MWAIT}}$ (Active Low) will put the CPU on the board into wait states providing additional time for the iSBX Multimodule board to perform the requested operation. $\overline{\text{MWAIT}}$ must be generated from address

(address plus chip select) information only. If $\overline{MWAIT}$ is driven active due to a glitch on the CS line during address transitions, $\overline{MWAIT}$ must be driven inactive in less than 75 ns.

The iSBX Multimodule board present ($\overline{MPST}$) is an active low signal (tied to signal ground) that informs the base board I/O decode logic that an iSBX Multimodule board has been installed.

Address and Chip Select Lines

The address and chip select lines are made up of two groups of signals.

Adress Lines: MA0–MA2

Chip Select Lines: $\overline{MCS0}$–$\overline{MCS1}$

The base board decodes I/O addresses and generates the chip selects for the iSBX Multimodule boards. The base board decodes all but the lower order three addresses in generating the iSBX Multimodule board chip selects.

Address Lines (MA0–MA2)

These positive true input lines to the iSBX Multimodule boards are generally the least three significant bits of the I/O address. In conjunction with the command and chip select lines, they establish the I/O port address being accessed. In 16-bit systems, MA0–MA2 may be connected to ADR1–ADR3 of the base board address lines.

Chip Select Lines (MCS0–MCS1/)

In an 8-bit system, these input lines to the iSBX Multimodule board are the result of the base board I/O decode logic. $\overline{MCS}$ is an active low signal which conditions the I/O command signals and thus enables communication with the iSBX Multimodule boards.

The SBX82062 Design Example

The SBX82062 Multimodule board will interface an ST506 compatible drive to any host board having an SBX connector. Two restrictions on the disk drive are that there is a maximum of 1024 cylinders and/or 8 heads. The SBX connector cannot supply the power-up current requirements of the drive. The drive must be connected directly to the power supply. The SBX82062 in Appendix C does not support DMA transfers. The version in Appendix D does support DMA transfers. Since this multimodule has a 2 kbyte sector buffer, the host microprocessor must generate a BRDY by accessing an I/O port during data transfers.

The software for communicating to the SBX board is intended to be interrupt driven. Polling for data transfers is not supported. Reading the status without an interupt is not recommended. During the times the 82062 is accessing the sector buffer, the SBX82062 will isolate itself from the host. To support polling, a hardware generated busy pattern should be driven onto the hosts's data bus as is shown in the Polled Interface section. The sector buffer stores up to 2 kbytes of disk data, for multiple sector transfers. The SBX board only interfaces to one drive (for space reasons), but four drives could be used with the addition of a read data multiplexor (one IC) and the drive data cables.

Microprocessor Interface

Figure 29 is a block diagram of the SBX82062's microprocessor interface. The I/O port assignments are listed in Table 1. The functional blocks of the interface are:

Sector Buffer Isolation Logic

Wait State Logic

Sector Buffer

Sector/Drive/Head Register Logic

Table 6-1. I/O Port Assignments

Port Address	Read	Write
80H	Sector Buffer	Sector Buffer
82H	Error Reg	RWC Reg
84H	Sector Count	Sector Count
86H	Sector Number	Sector Number
88H	Cylinder Low	Cylinder Low
8AH	Cylinder High	Cylinder High
8CH	SDH Reg	SDH Reg
8EH	Status Reg	Command Reg
90H	None	None
92H	None	Asserts $\overline{BCR}$
94H	None	Asserts BRDY

NOTE:
Address assignments are determined by the host board.

Sector Buffer Isolation Logic

The host will be isolated from the SBX board whenever the 82062 is accessing its sector buffer which is enabled by $\overline{BCS}$. The host's control signals, $\overline{RD}$, $\overline{WR}$, $\overline{MCS0}$, and $\overline{MCS1}$ and data bus are also disabled at the same time to prevent any data in the sector buffer from being corrupted. The host should wait for an interrupt before reading the 82062's Status register. Attempting to read the SBX board while $\overline{BCS}$ is active will return invalid data, since the SBX board will have the data bus tristated.

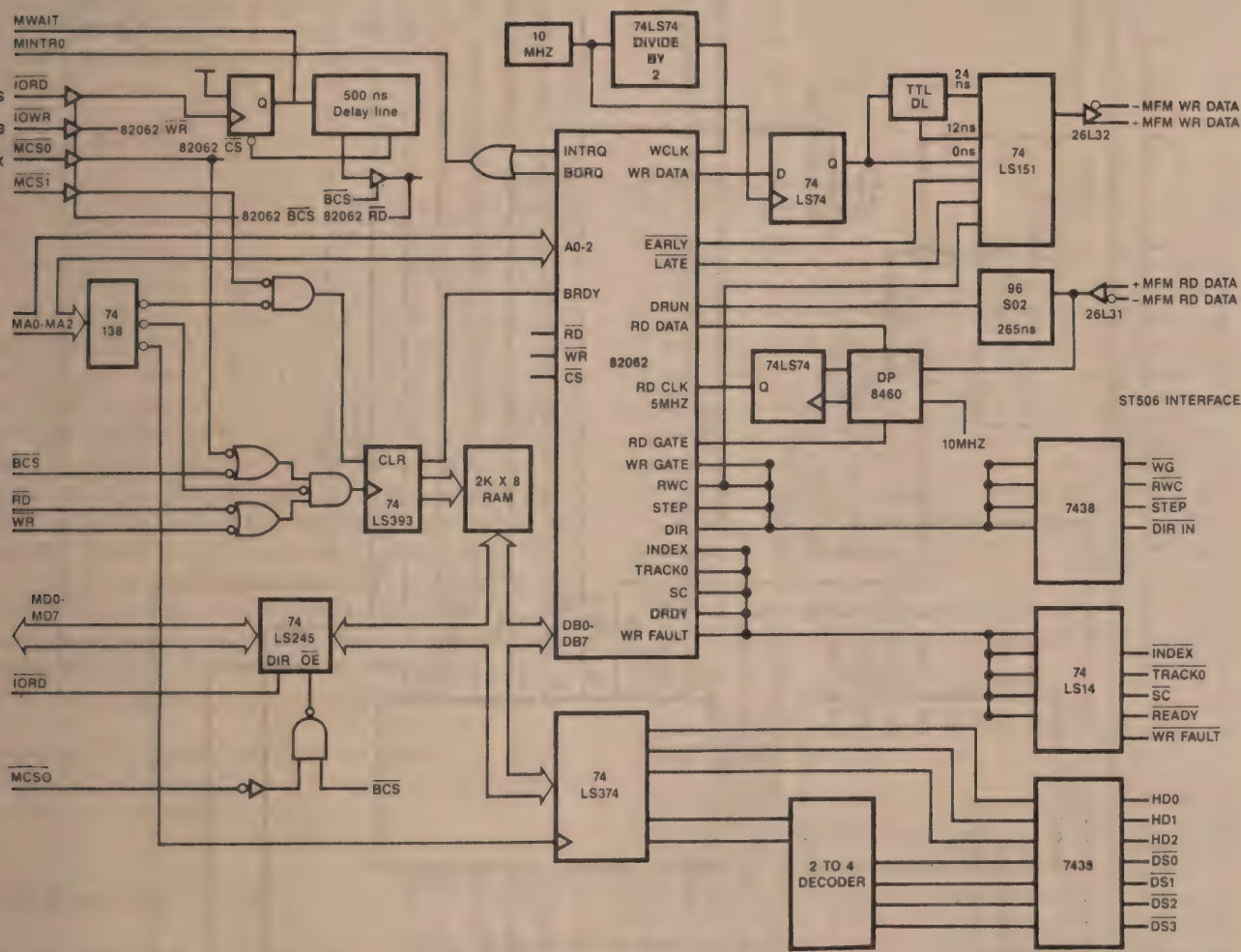

MWAIT
MINTR0
IORD
IOWR
MCS0
MCS1
S
B
X
82062 WR
82062 CS
Q
500 ns Delay line
BCS
82062 BCS
82062 RD
10 MHZ
74LS74 DIVIDE BY 2
INTRQ
BORD
WCLK
WR DATA
A0-2
EARLY
LATE
BRDY
DRUN
RD DATA
RD
WR
CS
82062
RD CLK 5MHZ
RD GATE
WR GATE
RWC
STEP
DIR
INDEX
TRACK0
SC
DRDY
WR FAULT
DB0-DB7
D
Q
74 LS74
TTL DL
24 ns
12ns
0ns
74 LS151
26L32
- MFM WR DATA
- MFM WR DATA
96 S02
265ns
26L31
+ MFM RD DATA
- MFM RD DATA
74LS74
Q
DP 8460
10MHZ
ST506 INTERFACE
MA0-MA2
74 138
BCS
RD
WR
CLR
74 LS393
2K X 8 RAM
7438
WG
RWC
STEP
DIR IN
MD0-MD7
74 LS245
DIR
OE
IORD
74 LS14
INDEX
TRACK0
SC
READY
WR FAULT
MCS0
BCS
74 LS374
2 TO 4 DECODER
7438
HD0
HD1
HD2
DS0
DS1
DS2
DS3

Figure 29

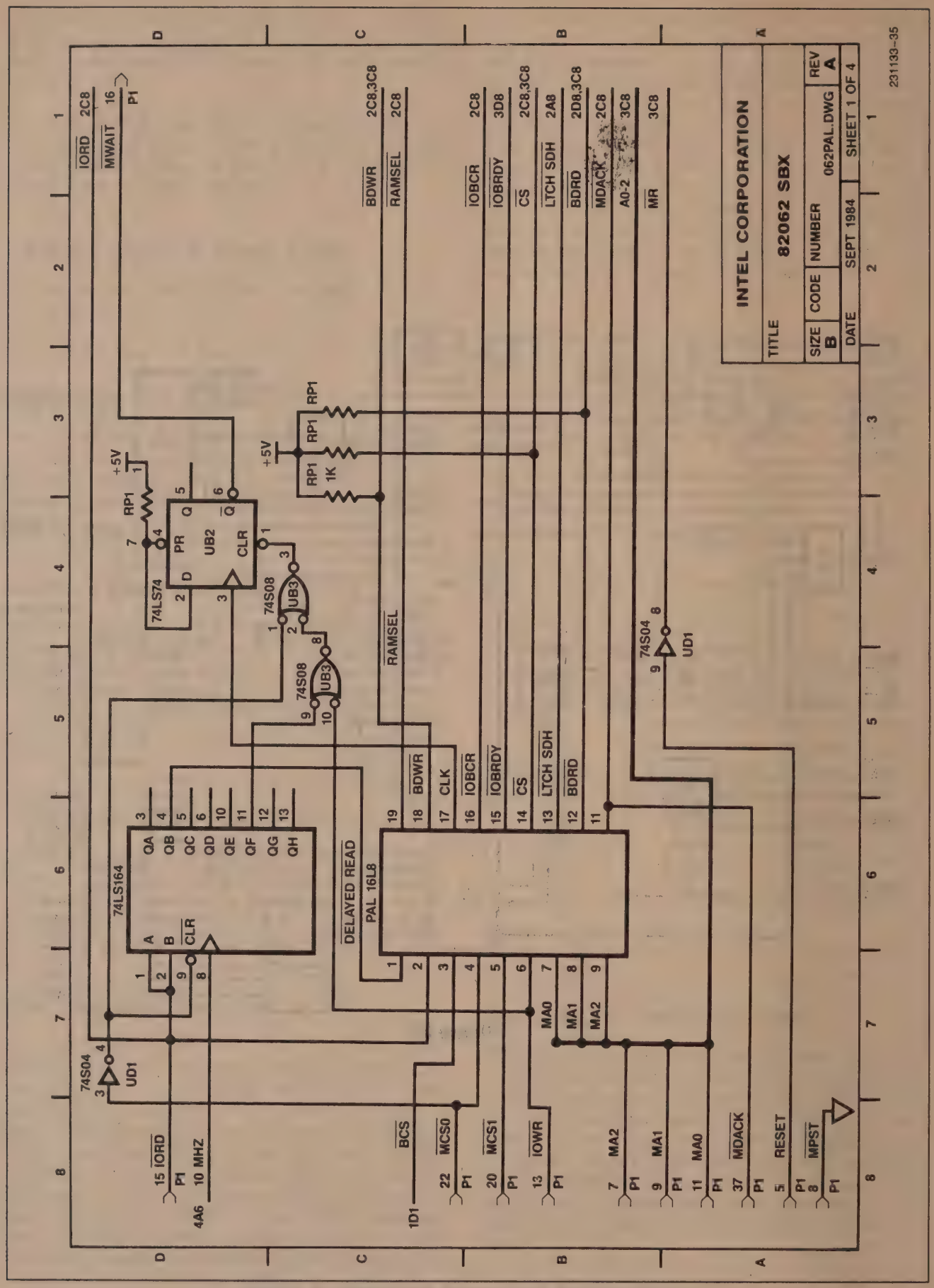


Figure 29. 82062 SBX Multimodule Block Diagram

231133-002

Wait State Logic

The wait state logic drives the 'not ready' line, $\overline{MWAIT}$, active whenever the host reads the SBX board. $\overline{MWAIT}$ does not go active for buffer or 82062 register writes. This logic was required for two reasons. First, a delayed read is generated, because the address setup to $\overline{RD}$ margin of the SBX bus is less than the 82062's needs (50 ns vs 100 ns). Second, the $\overline{RD}$ to data valid access period of the 82062 (375 ns), is greater than the SBX bus' full speed read cycle (275 ns) permits. $\overline{MWAIT}$ is deactivated after allowing for the delayed $\overline{RD}$ and the access period of the 82062. This delay is accomplished with a 500 ns delay line. The first tap at 100 ns generates the read request to allow for the address setup margin. The next tap 400 ns later removes $\overline{MWAIT}$ to allow the host to continue.

Sector Buffer

The sector buffer consists of an address counter (using '1s393's) and a 2 kbyte static RAM. The address counter is incremented on the trailing edge of a valid $\overline{RD}$ or $\overline{WR}$ cycle, either host microprocessor or 82062 initiated. The counter is reset by a hardware reset, the 82062 buffer reset $\overline{BCR}$, or by accessing an I/O port to provide software control. The 82062 will issue $\overline{BCR}$ each time $\overline{BCS}$ changes state (i.e. twice per sector). Resetting the buffer counter can be put under software control for multiple sector transfers. BRDY going high tells the 82062 that the buffer is available for its use. BRDY is generated by the address counter, by filling or emptying the entire buffer in multiple sector transfers, or from an I/O port when single sector transfers are done (since single sectors won't use all 2 kbytes of the buffer, the hardware signal will not be generated). When the 82062 is using the buffer, $\overline{BCS}$ will be low, and the $\overline{RD}$ or $\overline{WR}$ line will be pulsed every 1.6 microseconds.

When the 82062 is using the buffer it prevents access by the host by tristating the read, write, select and data lines with a low on $\overline{BCS}$.

SDH Register Logic

The drive and head select bits must be latched externally to the 82062, since these outputs are not provided. An 8 bit latch is strobed on the trailing edge of the $\overline{WR}$ pulse when the SDH register is selected. The two drive select bits are then demultiplexed to provide a one of four drive select line. If multiple drives are used then these outputs would also be used to select which disk's read data line would be gated into the PLL.

Interrupts

While the interrupt line is programmable (to notify of an end of command or data transfer request for the Read Sector command only), software will ensure that the interrupt from the 82062 signifies command termination. The BDRQ line is OR'ed with the 82062's INTRQ line or BDRQ can generate its own interrupt. BDRQ is also gated off-board for a DMA controller.

Disk Interface

Figure 30 is a block diagram of the interface between the 82062 and the disk drive. The functional blocks are:

Write Data Logic

Read Data Logic (PLL)

Drive Control

Write Data Logic

The WR DATA output requires a D flip-flop clocked at 10 MHz to complete the conversion of data to MFM. The output of this D flip-flop is true MFM and is sent to a delay line. A delay line determines the amount of delay for precompensation. No delay corresponds to shifting the data bit early; the first tap is approximately 12 ns of delay and is the "normal", or no delay and the second tap provides 12 ns of delay, referenced to the "normal" write data. Which output is selected is determined by the states on RWC, $\overline{Early}$ and $\overline{Late}$. This function was generated with a 74s151 multiplexer. When RWC is inactive $\overline{EARLY}$ and $\overline{LATE}$ only select "normal" data since they are always active. The precompensated write data is then driven onto the data cable by an RS-422 driver.

Read Data Logic

The PLL generates the RD CLOCK that is used to decode the serial MFM data from the drive. A selected drive issues read data, unless WR GATE is active. A one-shot generates a pulse of 220–270 ns to provide the DRUN input. Only during an all zero's or one's field will the DRUN input stay high, as it will be retriggered every 200 ns (the minimum distance that separates continuous clock and data bits). As soon as DRUN is determined to be valid, the RD GATE output will go active, switching the PLL from the 10 MHz local clock input to disk data. The PLL will synchronize to the incoming serial data and generate a Read Clock of the proper timing and phase. The 82062 will then start to search for the address mark which is indicated by DRUN going low at the address mark.

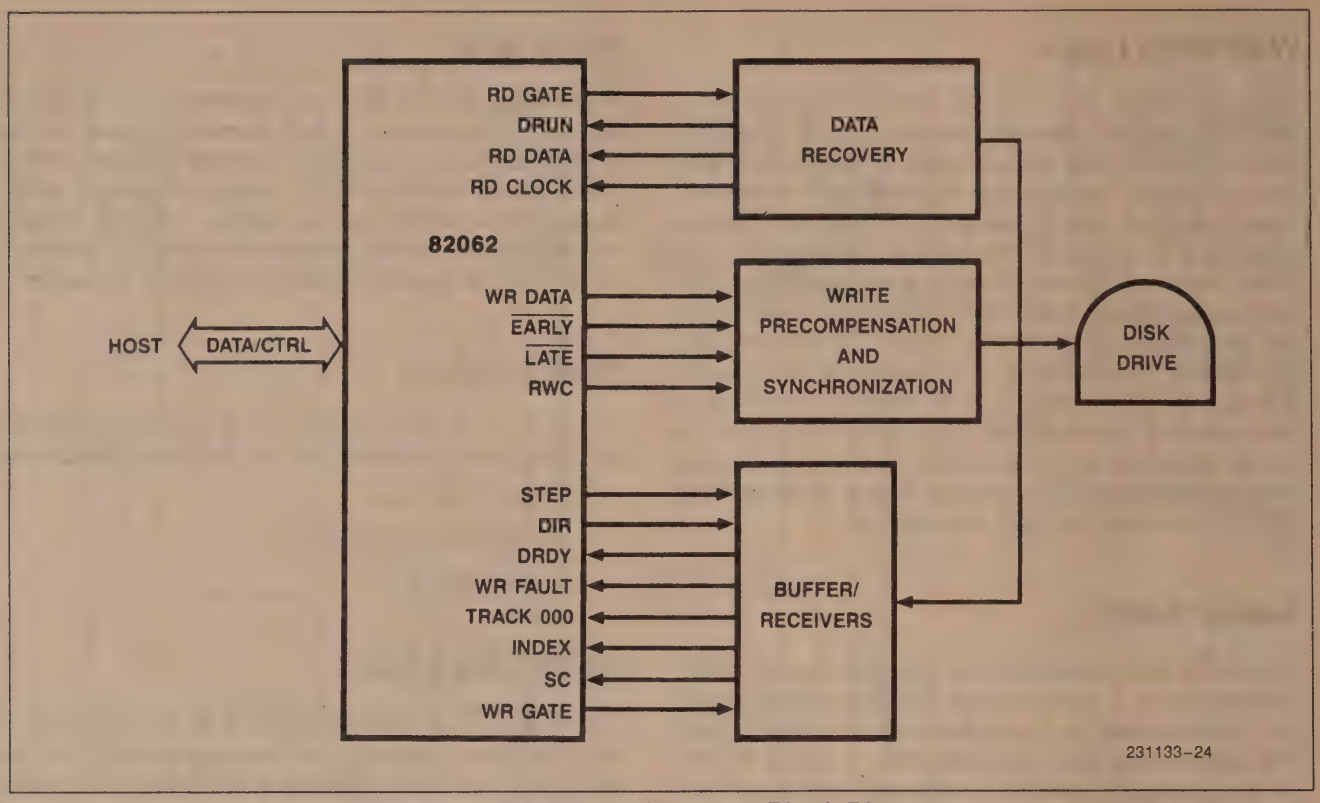


Figure 30. 82062 Disk Interface Block Diagram

No detail is provided herein on PLL design, as it is beyond the scope of this document. PLL design should be left to experienced designers, since minute changes in temperature and component values will drastically affect the soft error rate. As an alternative, several companies manufacture very high speed PLL chips for MFM encoded disk drives. Besides being fairly easy to design in, they reduce the number of components and board area needed for the sophisticated PLL.

Software Driver Overview

Presented in Appendix B is a listing of the software used to exercise the SBX 82062 board. Communication between the host software and the SBX driver routine is done through a structure located in system RAM. The host routine fills in required parameters, then passes the address of this communication block to the SBX driver routine. The driver routine pulls necessary values from this command block (CBL), executes a disk operation, then fills the CBL with the 82062's register contents, plus status and error information. The command block structure is shown in Figure 31.

Command	Byte
Rwc Reg	Byte
Sector Cnt.	Byte
Sector Num.	Byte
Cyl Low	Byte
Cyl High	Byte
SDH Reg	Byte
Status Reg	Byte
Error Reg	Byte
Host Buffer	Pointer

Figure 31

The host board did not have a DMA controller available, so an interrupt is issued from the BDRQ line and OR'ed with the 82062's interrupt line as interrupt sources were limited by the host. When an interrupt occurs, the interrupt procedure checks for either a data transfer, and executes it, or the completion of the command. If the interrupt signifies command completion, the interrupt procedure fills the command block with the 82062's task, status and error registers.

In this example, the host software examines one byte in the command block and until this byte is changed to a 00, no other command blocks will be passed to the disk driver routine. An alternative would be to issue a software interrupt to notify the microprocessor that the disk operation has finished and the command block contains parameters from the last operation and that a new disk command could start.

The driver for this example allows polling for non-data transfer commands, and must use interrupts for data transfers. As mentioned earlier, microprocessor intervention is required since the sector buffer is much larger than one sector and will not generate a BRDY. The microprocessor must write to an I/O port, which sets BRDY, after each host to sector buffer transfer. An actual software implementation would not include the polling and interrupt routines together, as only one method would generally be used.

The calling routine, which would normally be a directory program, places the values for which sector, number of sectors, etc., in the CBL. The disk routine is called and the address of this structure is passed on the stack. The disk driver places these parameters in the 82062's Task registers and initiates a command.

If the interrupt driven method was chosen, the disk driver routine returns to the calling routine. This permits other processing to be performed while the disk is executing a command. At some point, an interrupt will be generated, either from BRDY or INTRQ. Control will pass to the driver and the status register will be checked. If a data transfer is needed, either the microprocessor can transfer data or a DMA controller can perform the function. Once the transfer of data to the buffer is finished the microprocessor must set BRDY through an I/O port.

APPENDIX A
ST506 INTERFACE

THE ST506 INTERFACE

The ST506 interface is a modified version of Shugarts floppy disk drive interface and has been promoted by Seagate Technology. This interface is intended to be easy and low in cost to implement, yet provide a medium level of performance. The interface rigidly defines several areas: the hardware interconnects, the data transfer rate, the data encoding method, and how the disk is formatted.

Data Transfer Rate

The data transfer rate depends upon the linear bit density of the disk media and the speed at which the disk spins. ST506 specifies a 5 Mbit/second transfer rate. The typical ST506 drive has a nominal linear density of 10,416 bytes and a disk speed of 3600 rpm, which yields a 5 Mbit/second data transfer rate. No deviation from 5 M/bits second is allowed.

Increasing the linear density to increase storage capacity would require a decrease in disk speed. Otherwise, the data rate would increase. This decrease in disk speed would cause access times to increase, which many would deem unacceptable. To increase storage capacity, and remain ST506 compatible, either the number of cylinders and/or the number of platters can increase.

Data Encoding

ST506 requires that the serial data, sent between the drive and the controller, be encoded according to MFM rules. The basic unit of storage is a bit cell, which stores one bit infromation. This bit cell is divided into two halves, consisting of a clock bit and a data bit (see Figure A-1).

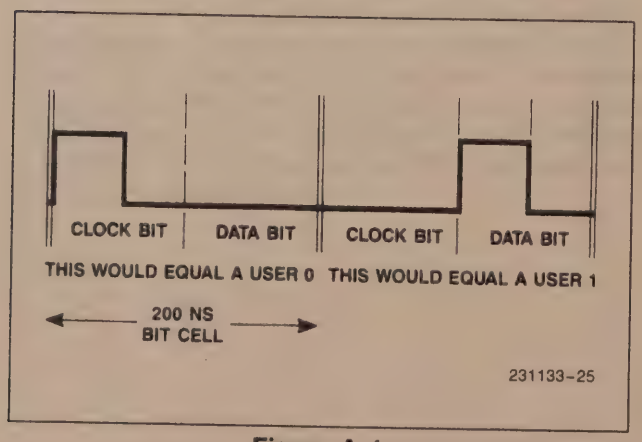


Figure A-1

The encoding rules for MFM are fairly simple:

1. A clock bit is written when the previous and the current bit cell does not contain a data bit.

2. A data bit is written whenever there is a "one" from the user.

Sync fields are composed of zeroes which generates a series of clock bits in the bit cell's. A phase lock loop locks on to the data stream during this period and generates a signal of the proper phase and frequency which is used to decode the combined clock and data serial data stream.

Disk Format

All disk media must be written with a specified format so that data may be reliably stored and retrieved. The smallest unit of controller accessible data is the sector which typically contains sync fields, ID fields, and a data field, and buffer fields.

The format of the disk required by ST506 is shown in Figure A-2. It should be noted that this format is fixed in the 82062. The user has options only for GAP1 and 3 length (when changing sector size or ECC) and whether to have 82062 CRC checking or user supplied ECC syndrome bits.

Gap 1 – Index Gap

Gap 1 serves two purposes. The first is to allow for variations in the index pulse timing due to motor speed variations. The second purpose is to allow a small delay to permit a different head to be selected without missing a sector. This is more of a data transfer optimization function and requires the disk controller to know which head is to be selected, when the last sector of a track has been read, and the next logical sector in the file exists on another platter. The 82062 does not switch heads automatically. Whether this scheme can be used or not depends upon the μP being able to alter one register in the 82062, before the next sector passes beneath the heads.

This gap is typically 12 bytes long and is written by the 82062 as 4E Hex.

Gap 2 – Write Splice Gap

This gap follows the CRC bytes of the ID field and continues up to the data field address mark. When up-

dating a previously written sector, motor speed variations could turn on the write coil, as the head was passing over the ID field. This gap prevents this from occurring. The value written is OOH and also serves as the PLL sync field for the data field. The minimum value is determined by the "lock up" performance of the PLL. The 82062 writes sixteen bytes for this field once WG is activated. The user has no control over this field.

Gap 3 – Post Data Field Gap

Gap 3 is very similar to Gap 2 as it is used as a speed tolerance buffer also. Without this gap, and with the motor speed varying slightly, it would be possible for the upcoming sector's sync field and ID field to be overwritten. This value is '4E' H and is typically 15 bytes long. The 82062's Gap 3 length is programmable. The exact value is dependent upon several factors. Refer to 82062 Format command, Software Section: General Programming Section.

Gap 4 – Track Buffer Gap

This gap follows the last sector on a track and is written until an index pulse is received. Its purpose is to prevent the last sector from overflowing past the index gap, and absorb track length variations when ECC is used (ECC uses more bytes than CRC). The value is '4E' H and is about 320 bytes when CRC and 256 byte sectors are used. The 82062 writes this field only during formatting. The user has no control over the number of bytes written with the 82062.

ID Fields

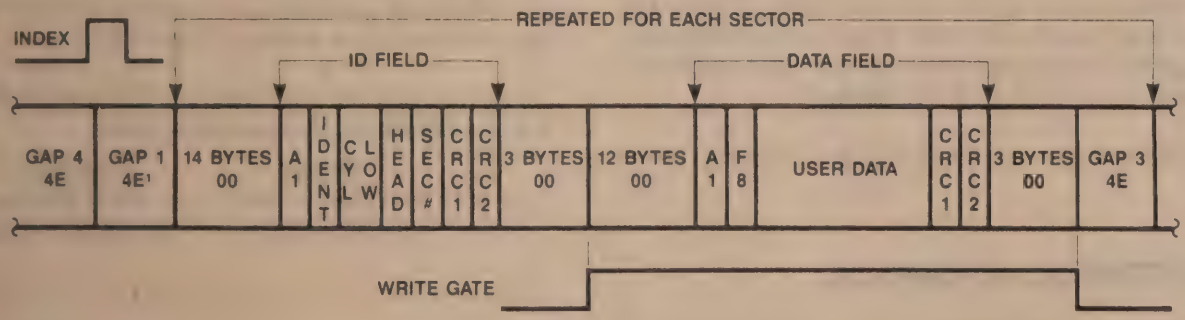
The controller uses ID fields to locate any individual sector. An address mark of two bytes precedes the ID field and the data field in a sector. An address mark tells the controller the nature of the upcoming information. ID fields are used by the disk controller and are not passed to the host.

Sector Interleaving

Sector interleaving occurs when logical sectors are in a non-sequential order, which is determined during formatting. An advantage is that there is a delay between logically sequential sectors. This delay can be used for data processing and then deciding if the next sector should be read. Without interleaving, the next sector could slip by, imposing a one revolution delay (approx. 16.7 ms). An additional benefit to this delay is that bus utilization is reduced by spreading the data transfer over a greater amount of time. The delay between sectors can be determined as follows:

$$\frac{\text{1 Revolution Period}}{\text{Sectors/Track}} \times (\text{Interleave factor} - 1) = \text{Delay}$$

For the typical ST506 drive with four-way interleaving this yields 1.57 ms of delay.



ID FIELD

A1 = A1 HEX WITH 0A HEX CLOCK
IDENT = 2LS.B. = CYLINDER HIGH
FE = 0-255 CYLINDERS
FF = 256-511 CYLINDERS
FC = 512-767 CYLINDERS
FD = 768-1023 CYLINDERS

HEAD = BITS 0, 1, 2 = HEAD NUMBER
BITS 3, 4 = 0
BITS 5, 6 = SECTOR SIZE
BIT 7 = BAD BLOCK MARK
SEC # = LOGICAL SECTOR NUMBER

DATA FIELD

A1 = A1 HEX WITH OA HEX CLOCK
F8 = DATA ADDRESS MARK; NORMAL CLOCK
USER = DATA FIELD 128 TO 1024 BYTES

231133-26

Figure A-2. Format Field

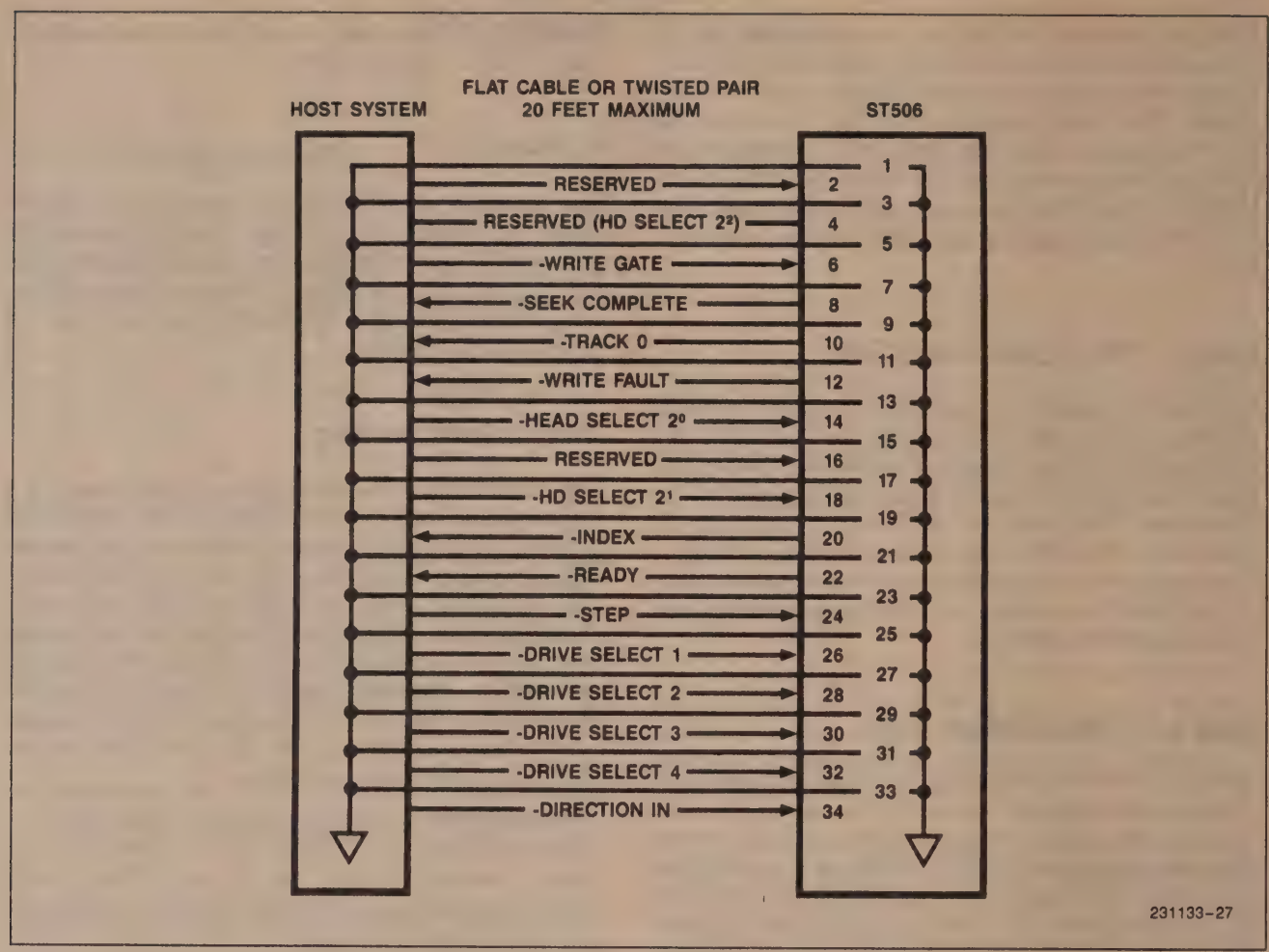


Figure A-3

The disadvantage to interleaving is that file transfers take longer, which may slow down the overall system. A four-way interleaved disk will have the transfer rate reduced to an average of 1.25 Mbit/sec.

The 82062 leaves the logical sector sequence to the user.

ELECTRICAL INTERFACE

The interface to the ST506 drive is divided into three categories and they are:

1. control signals,
2. data signals,
3. power.

Control Signals

The functions of the control signals are not covered in detail here. Their purpose can be found in the pin descriptions section. All control lines are digital in nature and either provide signals to the drive or inform the host of certain conditions. A diagram of the 34 pin control connector is shown in Figure A-3.

The driver/receiver logic diagram is shown in Figure A-4 and the electrical characteristics are:

	Voltage	Current
True	0.0 VDC to 0.4 VDC	−40 mA (IOL max.)
False	2,5 VDC to 5.25 VDC	250 μA (IOH open)

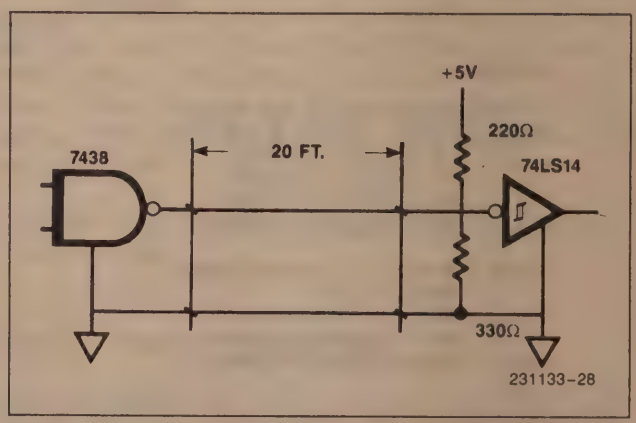


Figure A-4

Data Signals

The lines associated with the transfer of read/write data between the host and the drive are differential in nature and may not be multiplexed between drives. There is one pair of balanced lines for each read and write data line per drive and must conform to the RS-422 specification. Figure A-5 shows the receiver/transmitter combination.

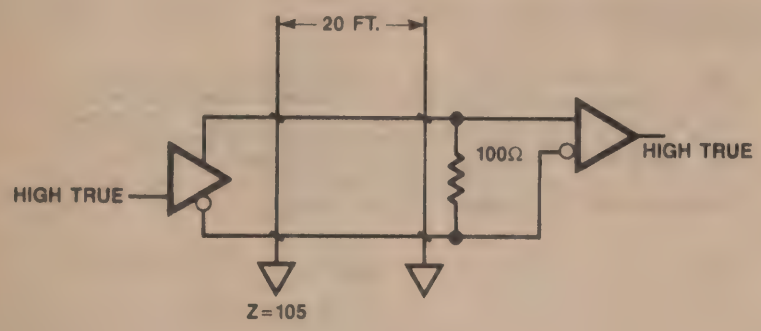


Figure A-5. E1A RS22 Driver/Receiver Pair Flat Ribbon or Twisted Pair

APPENDIX B
SOFTWARE DRIVER

```
SERIES-III PL/M-86 V2.3 COMPILATION OF MODULE DISK_IO_MODULE
OBJECT MODULE PLACED IN :F2:DISKIO.OBJ
COMPILER INVOKED BY:   PLM86.86 :F2:DISKIO.P86


                $TITLE('82062/SBX DISK CONTROLLER')
   1            DISK_IO_MODULE:DO;

                /*  CBL_PTR IS A POINTER TO A COMMAND BLOCK- HENCE CBL.
                    THIS COMMAND BLOCK RESIDES IN RAM AND CONTAINS ALL
                    VALUES REQUIRED BY THIS PROGRAM TO OPERATE THE 82062
                    DISK CONTROLLER. ONCE THIS PROCEDURE IS CALLED, THE
                    CBL IS REMAIN UNTOUCHED UNTIL THE COMMAND BYTE IS
                    SET TO A 00 VALUE. THIS ROUTINE WILL CALL THE CALLING
                    PROGRAM WHEN A COMMAND IS COMPLETED.

                    REV            DATE        NAME        DESCRIPTION

                    1.0          1/JUL/84   J. SLEEZER     INITIAL
                */


                /*        PROGRAM CONSTANTS                  */

   2    1       DECLARE LIT              LITERALLY  'LITERALLY',
                        TRUE             LIT      '0FFH',
                        FALSE            LIT      '00H',
                        FOREVER          LIT      'WHILE TRUE';


                /*        BOARD ADDRESSING    FOR THE 86/05        */

   3    1       DECLARE BASE_ADDR        LIT      '80H',
                        SCTR_BFFR        LIT      'BASE_ADDR',
                        ERR_REG          LIT      'BASE_ADDR + 02H',   /* READ ONLY */
                        SEC_CNT_REG      LIT      'BASE_ADDR + 04H',
                        SEC_NUM_REG      LIT      'BASE_ADDR + 06H',
                        CYL_LOW_REG      LIT      'BASE_ADDR + 08H',
                        CYL_HI_REG       LIT      'BASE_ADDR + 0AH',
                        S_DR_HD_REG      LIT      'BASE_ADDR + 0CH',
                        STATUS_REG       LIT      'BASE_ADDR + 0EH',   /* READ ONLY */
                        COMMAND_REG      LIT      'BASE_ADDR + 0EH',   /* WRITE ONLY */
                        WR_PCMP_REG      LIT      'BASE_ADDR + 02H',   /* WRITE ONLY */
                        BFFR_RESET       LIT      '92H',
                        BFFR_RDY         LIT      '94H',
                        SEC_BUF          LIT      '2048';


                /*******        82062 COMMANDS       ********/

   4    1       DECLARE RESTORE          LIT      '1FH',
                        SEEK             LIT      '7FH',
                        FORMAT           LIT      '50H',
                        SCAN_ID          LIT      '40H',
                        READ_SEC         LIT      '20H',
                        WRITE_SEC        LIT      '30H',
                        ECC_EN           LIT      '80H',   /* TO BE OR'D WITH VALUE IN SDH REG */
                        NO_INTERPT       LIT      '00H',
                        INTR_ON_CMD      LIT      '08H',
                        MULT_SCTR        LIT      '04H';
```

231133-36

```
          /*      STATUS REGISTER BITS          */

 5   1    DECLARE ERR              LIT     '01H',
                  CIP              LIT     '02H',
                  DRQ              LIT     '80H',
                  SC               LIT     '10H',
                  WRF              LIT     '20H',
                  DRDY             LIT     '40H',
                  BUFBSY           LIT     '80H';  /* USER WILL NEVER SEE THIS BIT SET */

          /*      PROGRAM VARIABLES             */

 6   1    DECLARE CMD_BLOCK_PTR    POINTER,
                  CBL              BASED           CMD_BLOCK_PTR STRUCTURE (
                                   COMMAND         BYTE,
                                   PRECMP          BYTE,
                                   S_CNT           BYTE,
                                   SCTR            BYTE,
                                   LOW_CYL         BYTE,
                                   HI_CYL          BYTE,
                                   SDH             BYTE,
                                   STATUS          BYTE,
                                   ERRS            BYTE,
                                   INTERUPT        BYTE,
                                   RET_PROC        POINTER,
                                   BUFF_PTR        POINTER);

 7   1    DECLARE BUFFER_PTR       POINTER,
                  BUFF             BASED   BUFFER_PTR (1) BYTE,
                  STATUS           BYTE,
                  ERRORS           BYTE,
                  COMMAND          BYTE;


          $EJECT




          /************************************/

          /*      82062 POLL ROUTINE           */

          /************************************/

 8   1    POLL: PROCEDURE;

 9   2    DECLARE COUNT DWORD;

10   2        COUNT = 7FFFFH;     /* LOOP FAILSAFE - TWEAK AS REQUIRED */
11   2        STATUS = INPUT(STATUS_REG);
12   2        DO WHILE ((STATUS AND (CIP OR DRDY)) = (CIP OR DRDY));
13   3            IF COUNT = 00 THEN RETURN;
15   3            IF (STATUS AND DRQ) = DRQ THEN DO;
17   4                CALL XFER_DATA;
18   4            END;
19   3            STATUS = INPUT(STATUS_REG);
20   3            COUNT = COUNT - 1;
21   3        END;                /* IF THE ROUTINE EXPIRES DUE TO COUNT = 0, ALL DISK */
                                  /* REG VALUES IN THE CBL WILL CONTAIN THE STATUS REG */
                                  /* WHICH WILL = A BUSY PATTERN AND CBL COMMAND WILL  */
                                  /* CONTAIN 00, INDICATING THE COMMAND IS FINISHED    */
22   2    END POLL;

          /*********************************************************************/

          /*      TRANSFER DATA BETWEEN HOST RAM AND ONBOARD SECTOR BUFFER    */

          /*********************************************************************/

23   1    XFER_DATA: PROCEDURE;

24   2    DECLARE CNT              BYTE,
                  INDEX            WORD,
                  SZ1              BYTE,
                  SECTR_SZ         WORD;
```

231133-37

```
25   2          SZ1 = (SHR(CBL.SDH,5) AND 03H);      /* OBTAIN SECTOR SIZE BITS  FROM SDH */
26   2          IF SZ1 = 00 THEN SECTR_SZ = 256;     /* REGISTER */
28   2          ELSE IF SZ1 = 01 THEN SECTR_SZ = 512;
30   2          ELSE IF SZ1 = 02 THEN SECTR_SZ = 1024;
32   2          ELSE IF SZ1 = 03 THEN SECTR_SZ = 128;

34   2          IF CBL.SDH AND ECC_EN = ECC_EN THEN
35   2               SECTR_SZ = SECTR_SZ + 7;

36   2          IF(((CBL.COMMAND AND 0F0H = READ_SEC) OR(CBL.COMMAND AND 0F0H = WRITE_SEC))
37   2              AND (CBL.COMMAND AND 0FH = MULT_SCTR)) THEN DO;/* VARIOUS SECTOR SIZES*/
38   3              CNT = (SEC_BUF/CBL.S_CNT);                    /* ARE POSSIBLE.  THIS FIGURES  */
39   3              DO WHILE (CNT * SECTR_SZ) > SEC_BUF;          /* HOW MANY SECTORS WILL FIT    */
40   4                  CNT = CNT - 1;                            /* INTO THE BOARDS SECTOR BFFR */
41   4              END;
42   3              SECTR_SZ = SECTR_SZ * CNT;
43   3          END;

           /*  OUTPUT(BFFR_RESET) = 00;  */

44   2          IF (SHR(CBL.COMMAND,4) AND 03H) = 02H THEN DO;   /*  READ COMMAND */
46   3              DO INDEX = 0 TO (SECTR_SZ - 1);
47   4                  BUFF(INDEX) = INPUT(SCTR_BFFR);
48   4              END;
49   3          END;
50   2          ELSE DO;                                  /*  WRITE  OR FORMAT COMMAND */
51   3              DO INDEX = 0 TO (SECTR_SZ - 1);
52   4                  OUTPUT(SCTR_BFFR) = BUFF(INDEX);
53   4              END;
54   3          END;

55   2          OUTPUT(BFFR_RDY) = 00;  /* ACTIVATES 062'S BRDY LINE */

56   2      END XFER_DATA;

           $EJECT




           /*************************************/

           /*      UPDATE COMMAND BLOCK          */

           /*************************************/

57   1      UPDATE_CBL: PROCEDURE;

58   2          CBL.S_CNT = INPUT(SEC_CNT_REG);
59   2          CBL.SCTR = INPUT(SEC_NUM_REG);
60   2          CBL.LOW_CYL = INPUT(CYL_LOW_REG);
61   2          CBL.HI_CYL = INPUT(CYL_HI_REG);
62   2          CBL.SDH = INPUT(S_DR_HD_REG);
63   2          CBL.STATUS = STATUS;
64   2          CBL.ERRS = INPUT(ERR_REG);

65   2      END UPDATE_CBL;

           /*************************************/

           /*      WRITE THE CBL TO 82062        */

           /*************************************/

66   1      WR_CBL: PROCEDURE;

67   2          OUTPUT(WR_PCMP_REG) = CBL.PRECMP;
68   2          OUTPUT(SEC_CNT_REG) = CBL.S_CNT;
69   2          OUTPUT(SEC_NUM_REG) = CBL.SCTR;
70   2          OUTPUT(CYL_LOW_REG) = CBL.LOW_CYL;
71   2          OUTPUT(CYL_HI_REG) = CBL.HI_CYL;
72   2          OUTPUT(S_DR_HD_REG) = CBL.SDH;

73   2      END WR_CBL;

           $EJECT
```

231133-38

```
                /*******************************************/
                /*****    MAIN PROGRAM          ***********/

 74   1         DISK: PROCEDURE(CBL_PTR) PUBLIC;

 75   2         DECLARE CBL_PTR POINTER;

 76   2             CMD_BLOCK_PTR = CBL_PTR;                        /* ADDRESS OF STRUCTURE */
 77   2             BUFFER_PTR = CBL.BUFF_PTR;                      /* THAT CONTAINS 82062   */
                                                                    /* TASK REG DATA  */

 78   2             CALL WR_CBL;                                    /* A DUMMY COMMAND TO READ*/
 79   2             IF CBL.COMMAND = 99H THEN DO;                   /*THE CURRENT REG VALUES  */
 81   3                 CALL UPDATE_CBL;
 82   3                 CBL.COMMAND = 00;
 83   3                 CBL.STATUS = INPUT(STATUS_REG);
 84   3                 RETURN;
 85   3             END;

 86   2             IF (INPUT(STATUS_REG) AND DRDY) <> DRDY THEN DO; /* NO COMMAND IS ISSUED */
 88   3                 CBL.STATUS = INPUT(STATUS_REG);              /* IF THE 82062 SEES    */
 89   3                 CBL COMMAND = 00H;                           /* THAT THE SELECTED    */
 90   3                 RETURN;                                      /* DRIVE IS NOT READY   */
 91   3             END;

 92   2             OUTPUT(BFFR_RESET) = 00H;

 93   2             IF (CBL.COMMAND AND 0F0H) = READ_SEC THEN     /* FOR PROGRAM CONSISTENCY */
 94   2                 CBL.COMMAND = CBL.COMMAND OR INTR_ON_CMD; /* SET INTERUPT FOR COMMAND*/
                                                                 /* TERMINATION */

 95   2             OUTPUT(COMMAND_REG) = CBL.COMMAND;       /* A DELAY IS NEEDED BECAUSE FAST*/
 96   2             CALL TIME(100);                          /* UP'S CAN READ THE STATUS REG  */
 97   2             IF CBL.INTERUPT = NO_INTERPT THEN DO;    /* BEFORE A VALID STATUS IS READY*/
 99   3                 CALL POLL;
100   3                 CALL UPDATE_CBL;
101   3                 CBL.COMMAND = 00;
102   3                 RETURN;
103   3             END;

104   2         END DISK;

                $EJECT


                /*******************************************/

                /*       INTERRUPT SERVICE ROUTINE          */

                /*******************************************/

105   1         DISK_SERVICE: PROCEDURE PUBLIC;

106   2             CALL TIME(500);
107   2             STATUS = INPUT(STATUS_REG);

108   2             IF (STATUS AND CIP) = 00  THEN DO;
110   3                 CALL UPDATE_CBL;
111   3                 CBL.COMMAND = 00;
112   3                 OUTPUT(BFFR_RESET) = 00H;
113   3                 RETURN;
114   3             END;
115   2             ELSE CALL XFER_DATA;

116   2         END DISK_SERVICE;

117   1         END DISK_IO_MODULE;

MODULE INFORMATION:

     CODE AREA SIZE      = 02EEH     750D
     CONSTANT AREA SIZE  = 0000H       0D
     VARIABLE AREA SIZE  = 0011H      17D
     MAXIMUM STACK SIZE  = 000AH      10D
     272 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS
```

231133-39

```
DICTIONARY SUMMARY:

     31KB MEMORY AVAILABLE
     5KB MEMORY USED    (16%)
     0KB DISK SPACE USED

END OF PL/M-86 COMPILATION
```

```
SERIES-III PL/M-86 V2.3 COMPILATION OF MODULE HOST_MODULE
OBJECT MODULE PLACED IN :F2:DSKHST.OBJ
COMPILER INVOKED BY:  PLM86.86 :F2:DSKHST.P86

                $TITLE('DEMO PROGRAM FOR SBX062')
  1             HOST_MODULE:DO;

                /*  PROGRAM TO EXERCISE THE 82062/SBX BOARD  USING THE 957 MONITOR
                 ON AN SBC 86/05.  THIS PROGRAM DEMONSTRATES HOW THE DISKIO MODULE
                 IS USED.  THE CASE STATEMENTS IN THE MAIN SECTION SHOW THE VARIOUS
                 ROUTINES.  THE TYPICAL ROUTINES LIKE HEX TO ASCII, ETC., WERE
                 NOT INCLUDED IN THIS LISTING.  SEVERAL OF THE ROUTINES USE
                 STATEMENTS THAT COULD BE REDUCED CONSIDERABLY BUT WERE LEFT
                 SIMPLIFIED SO THAT ALL WOULD UNDERSTAND.

                    REV            DATE         NAME        DESCRIPTION

                    1.0          20/JUL/84   J. SLEEZER     INITIAL
                */


                /*        EXTERNAL ROUTINES        */

  2   1         CO:  PROCEDURE(CHAR) EXTERNAL;
  3   2              DECLARE CHAR BYTE;
  4   2         END CO;


  5   1         CI:  PROCEDURE BYTE EXTERNAL;
  6   2         END CI;

  7   1         DISK:PROCEDURE(CMD_BLK_PTR) EXTERNAL;/* THIS ROUTINE STARTS A DISK OPERATION */
  8   2              DECLARE CMD_BLK_PTR POINTER;
  9   2         END DISK;

 10   1         DISK_SERVICE: PROCEDURE EXTERNAL;/* THIS ROUTINE SERVICES THE 82062 INTERUPTS*/
 11   2         END DISK_SERVICE;


               /*        PROGRAM CONSTANTS            */

 12   1        DECLARE LIT              LITERALLY  'LITERALLY',
                      TRUE             LIT      '0FFH',
                      FALSE            LIT      '00H',
                      FOREVER          LIT      'WHILE TRUE',
                      SPACE            LIT      '20H',
                      CR               LIT      '0DH',
                      LF               LIT      '0AH',
                      RUB              LIT      '7FH',
                      BACKSP           LIT      '08H',
                      ESC              LIT      '1BH';


               /*      82062 COMMANDS              */
```

231133-40

```
13   1     DECLARE RESTORE          LIT     '10H',
                 SEEK             LIT     '70H',
                 FORMAT           LIT     '50H',
                 SCAN_ID          LIT     '40H',
                 READ_SECT        LIT     '28H',   /* INTR ONLY ON COMMAND TERMINATION */
                 WRITE_SCT        LIT     '30H',
                 MULT_SCTR        LIT     '04H',   /* TO BE OR'D WITH COMMAND */
                 NO_RETRYS        LIT     '01H',   /* TO BE OR'D WITH COMMAND */
                 NO_CRC           LIT     '80H',   /* TO BE OR'D WITH VALUE IN SDH REG */

                 P_COMP           LIT     '0',   /*  INDEXING INTO DISK_REG ARRAY */
                 SEC_CNT          LIT     '1',
                 SECTOR           LIT     '2',
                 CYL_LB           LIT     '3',
                 CYL_HB           LIT     '4',
                 SDH              LIT     '5';

           /*      STATUS REGISTER BITS        */

14   1     DECLARE ERR              LIT     '01H',
                 CIP              LIT     '02H',
                 DRQ              LIT     '80H',
                 SC               LIT     '10H',
                 WRF              LIT     '20H',
                 DRDY             LIT     '40H',
                 BUFBSY           LIT     '80H';   /* USER WILL NEVER SEE THIS BIT SET */

           /*      ERROR REGISTER BITS         */

15   1     DECLARE VALID_BITS       LIT     '0D7H',
                 AM_NT_FND        LIT     '001H',
                 TK00_ERR         LIT     '002H',
                 ABRTD_CMD        LIT     '004H',
                 ID_NT_FND        LIT     '010H',
                 DATA_ERR         LIT     '040H',
                 BAD_BLK          LIT     '080H';

           /*****      PROGRAM VARIABLES       *****/

16   1     DECLARE CMD_BLK(1)       STRUCTURE ( COMMAND          BYTE,
                                                PRECMP           BYTE,
                                                S_CNT            BYTE,
                                                SCTR             BYTE,
                                                LOWB_CYL         BYTE,
                                                HIB_CYL          BYTE,
                                                SDHD             BYTE,
                                                STATUS           BYTE,
                                                ERRS             BYTE,
                                                INTERUPT         BYTE,
                                                RET_PROC         POINTER,
                                                BUFF_PTR         POINTER);

17   1     DECLARE COUNT                WORD,

                 CHAR                 BYTE,
                 ERRORS               BYTE,
                 COMMAND              BYTE,
                 STEP_RATE            BYTE,
                 I                    WORD,
                 I2                   BYTE,
                 BUFFER(1100)         BYTE,
                 INDEX                WORD,
                 DISK_IS_NOT_BUSY     BYTE,
                 TRACKS               BYTE,
                 PLATTERS             BYTE,
                 PLAT_CNT             BYTE,
                 TRACK_CNT            BYTE,
                 I_FACTOR             BYTE,
                 FRMT_BFFR_SIZE       BYTE,
                 LOG_SECT_NUM         BYTE,
                 MAKING_TABLE         BYTE,
                 AA                   BYTE,
                 INDX                 BYTE;

18   1     DECLARE DISK_REGS (6)        BYTE;

           $NOLIST
           $EJECT
```

231133-41

```
                /*******************************************/

                /*********      MAIN PROGRAM        *******/

                /*******************************************/

337   1         MAIN: DO;

338   2             STEP_RATE = 0FH;
339   2             PLAT_CNT =0FFH;
340   2             TRACK_CNT =0FFH;
341   2             PLATTERS = 00;
342   2             TRACKS = 00;
343   2             DO I = 0 TO 5;
344   3                 DISK_REGS(I) = 00;
345   3             END;
346   2             DISK_REGS(P_COMP) = 0FFH;
347   2             CALL UPDATE_CMD_BLK;
348   2             CMD_BLK(INDX).INTERUPT = 00;
349   2             CALL WRITEA(@(LF,LF,LF,LF,LF,00));
350   2             CALL WRITEA(@SIGN_ON);
351   2             OUTPUT(0C2H) = 0FDH;   /*  PERMITS AN INTERRUPT 1  */
352   2             INDX = 0;
353   2             CMD_BLK(INDX).BUFF_PTR = @BUFFER;
354   2             CALL SET$INTERRUPT(21H,CHECK_DISK);
355   2             ENABLE;
356   2             DISK_IS_NOT_BUSY = TRUE;

357   2             DO FOREVER;
358   3                 IF DISK_IS_NOT_BUSY THEN DO;
360   4                     CMD_BLK(INDX).COMMAND = 0FFH;
361   4                     CALL WRITEA(@(CR,LF,'COMMAND  >',00));
362   4                     CHAR = CI;
363   4                     CALL CO(CHAR);
364   4                     CALL CO(CR);
365   4                     CALL CO(LF);
366   4                     COMMAND = FALSE;
367   4                     I2 = 0;
368   4                     DO WHILE (COMMAND = FALSE);
369   5                         IF I2 > LENGTH(VALID_CMDS) THEN DO;
371   6                             CALL WRITEA(@('INVALID COMMAND',CR,LF,00));
372   6                             I2 = 0;
373   6                             CHAR = CI;
374   6                         END;


375   5                         IF CHAR = VALID_CMDS(I2) THEN
376   5                             COMMAND = TRUE;
377   5                         I2 = I2 + 1;
378   5                     END;
379   4                     DO CASE (I2 - 1);

                                /*   CASE 0 - READ SECTOR */
380   5                         DO;
381   6                             CALL WRITEA(@('READ SECTOR COMMAND',CR,LF,LF,00));
382   6                             CALL WRITE_REGS;
383   6                             DISK_IS_NOT_BUSY = FALSE;
384   6                             CMD_BLK(INDX).COMMAND = READ_SECT;
385   6                             CALL WRITEA(@('MULTIPLE SECTOR'S? >',00));
386   6                             CHAR = CI;
387   6                             IF CHAR = 'Y' THEN DO;
389   7                                 CALL WRITEA(@('YES - ',00));
390   7                                 CMD_BLK(INDX).COMMAND =
                                                CMD_BLK(INDX).COMMAND OR MULT_SCTR;
391   7                                 CALL WRITEA(@('DO NOT EXCEED BUFFER LIMIT !',CR,LF,00));
392   7                             END;
393   6                             ELSE CALL WRITEA(@('NO',CR,LF,00));

394   6                             CALL WRITEA(@('AUTOMATIC RETRIES? >',00));
395   6                             CHAR = CI;
396   6                             IF CHAR = 'N' THEN DO;
398   7                                 CALL WRITEA(@('NO',CR,LF,00));
399   7                                 CMD_BLK(INDX).COMMAND =
                                                CMD_BLK(INDX).COMMAND OR NO_RETRYS;
400   7                             END;
401   6                             ELSE CALL WRITEA(@('YES',CR,LF,00));

402   6                             CALL DISK(@CMD_BLK(INDX));
403   6                         END;
```

231133-42

```
                    /*   CASE 1 - WRITE SECTOR */
5                   DO;
6                       CALL WRITEA(@('WRITE SECTOR COMMAND',CR,LF,LF,00));
6                       CALL WRITE_REGS;
6                       CALL DATA_PAT;
6                       DISK_IS_NOT_BUSY = FALSE;
6                       CMD_BLK(INDX).COMMAND = WRITE_SCT;
6                       CALL WRITEA(@('MULTIPLE SECTOR'S? >',00));
6                       CHAR = CI;
6                       IF CHAR = 'Y' THEN DO;
7                           CALL WRITEA(@('YES - ',00));
7                           CMD_BLK(INDX).COMMAND =
                                    CMD_BLK(INDX).COMMAND OR MULT_SCTR;
7                           CALL WRITEA(@('DO NOT EXCEED BUFFER LIMIT !!',CR,LF,00));
7                       END;
6                       ELSE CALL WRITEA(@('NO',CR,LF,00));

6                       CALL WRITEA(@('ENABLE  RETRIES ? >',00));
6                       CHAR = CI;
6                       IF CHAR = 'N' THEN DO;
7                           CALL WRITEA(@('NO',CR,LF,00));
7                           CMD_BLK(INDX).COMMAND =
                                    CMD_BLK(INDX).COMMAND OR NO_RETRYS;
7                       END;
6                       ELSE CALL WRITEA(@('YES',CR,LF,00));

6                       CALL DISK(@CMD_BLK(INDX));
6                   END;

                    /*   CASE 2 - FORMAT TRACK */
5                   DO;
6                       CALL WRITEA(@('FORMAT TRACK',CR,LF,LF,00));
6                       CALL WRITE_REGS;
6                       DISK_IS_NOT_BUSY = FALSE;
6                       CMD_BLK(INDX).COMMAND = FORMAT;
6                       CALL WRITEA(@(' INTERLEAVE FACTOR? (1 TO ?)>',00));
6                       I_FACTOR = CI - '0';
6                       CALL CO(I_FACTOR + '0');
6                       CALL CO(CR);
6                       CALL CO(LF);
6                       FRMT_BFFR_SIZE = (2 * (CMD_BLK(INDX).S_CNT) + 1);
6                       DO I = 0 TO FRMT_BFFR_SIZE;
7                           BUFFER(I) = 00;
7                       END;
6                       LOG_SECT_NUM = 0;
6                       I= 1;
6                       MAKING_TABLE = TRUE;
6                       DO WHILE MAKING_TABLE;
7                           DO WHILE I <= FRMT_BFFR_SIZE;
8                               BUFFER(I) = LOG_SECT_NUM;
8                               LOG_SECT_NUM = LOG_SECT_NUM + 1;
8                               I = I +(I_FACTOR * 2);
8                           END;
7                           IF LOG_SECT_NUM < CMD_BLK(INDX).S_CNT THEN DO;
8                               I = I - (FRMT_BFFR_SIZE + 1);
8                               IF (I = 1) OR (BUFFER(I) <> 00) THEN
8                                   I = I + 2;
8                           END;
7                           ELSE MAKING_TABLE = FALSE;
7                       END;
6                       CALL WRITEA(@('256 TRACKS IS THE LIMIT',CR,LF,00));
6                       CALL WRITEA(@('HOW MANY TRACKS? IN HEX >',00));
6                       TRACKS = HEXIN(TRACKS);
6                       CALL CO(CR);
6                       CALL CO(LF);
6                       CALL WRITEA(@('HOW MANY SURFACES? I.E.,01 >',00));
6                       PLATTERS = HEXIN(PLATTERS);
6                       CALL CO(CR);
6                       CALL CO(LF);
6                       TRACK_CNT = 1;
```

231133-43

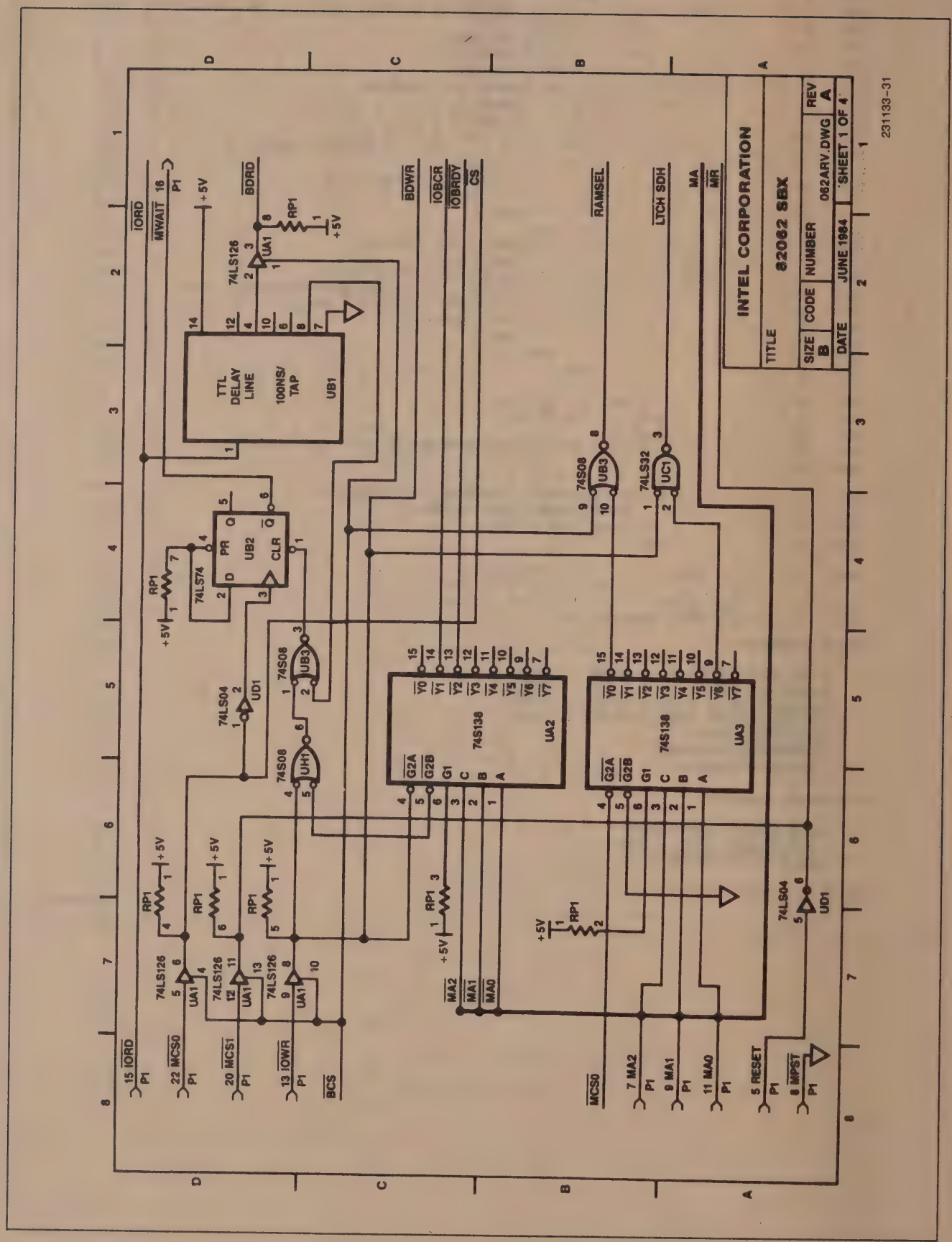
```
470  6                    DO WHILE TRACK_CNT <= TRACKS;
471  7                        PLAT_CNT = 1;
472  7                        DO WHILE PLAT_CNT <= PLATTERS;
473  8                            CALL UPDATE_CMD_BLK;
474  8                            CALL CO(CR);
475  8                            CALL WRITEA(@('TRACK = ',00));
476  8                            CALL DISP_HEX(@TRACK_CNT,1);
477  8                            CALL WRITEA(@('; HEAD = ',00));
478  8                            AA = DISK_REGS(SDH) AND 07H;
479  8                            CALL DISP_HEX(@AA,1);
480  8                            CMD_BLK(INDX).COMMAND = FORMAT;
481  8                            CALL DISK(@CMD_BLK(INDX));
482  8                            DO WHILE CMD_BLK(INDX).COMMAND <> 00;
483  9                            END;
484  8                            PLAT_CNT = PLAT_CNT + 1;
485  8                            DISK_REGS(SDH) = DISK_REGS(SDH) + 1;
486  8                        END;
487  7                        DISK_REGS(SDH) = DISK_REGS(SDH) - (PLATTERS);
488  7                        DISK_REGS(CYL_LB) = DISK_REGS(CYL_LB) + 1;
489  7                        IF DISK_REGS(CYL_LB) = 00 THEN
490  7                            DISK_REGS(CYL_HB) = DISK_REGS(CYL_HB) + 1;
491  7                        TRACK_CNT = TRACK_CNT + 1;
492  7                        CALL UPDATE_CMD_BLK;
493  7                    END;
494  6                    CALL CO(CR);
495  6                    CALL CO(LF);
496  6                END;

                      /*   CASE 3 - SCAN ID   */
497  5                DO;
498  6                    CALL WRITEA(@(' SCAN ID',CR,LF,LF,00));
499  6                    CALL WRITE_REGS;
500  6                    DISK_IS_NOT_BUSY = FALSE;
501  6                    CMD_BLK(INDX).COMMAND = SCAN_ID;
502  6                    CALL DISK(@CMD_BLK(INDX));
503  6                END;

                      /*   CASE 4 - SEEK TRACK */
504  5                DO;
505  6                    CALL WRITEA(@('SEEK TRACK',CR,LF,LF,00));
506  6                    CALL WRITE_REGS;
507  6                    CMD_BLK(INDX).COMMAND = SEEK OR STEP_RATE;
508  6                    DISK_IS_NOT_BUSY = FALSE;
509  6                    CALL DISK(@CMD_BLK(INDX));
510  6                END;
                      /*   CASE 5 - RESTORE */
511  5                DO;
512  6                    CALL WRITEA(@(' RESTORE COMMAND',CR,LF,LF,00));
513  6                    CALL WRITE_REGS;
514  6                    CMD_BLK(INDX).COMMAND = RESTORE OR STEP_RATE;
515  6                    DISK_IS_NOT_BUSY = FALSE;
516  6                    CALL DISK(@CMD_BLK(INDX));
517  6                END;

                      /*   CASE 6 - READ DISK REGISTER FILE */
518  5                DO;
519  6                    CALL WRITEA(@(' READ DISK REGISTERS',CR,LF,LF,00));
520  6                    CMD_BLK(INDX).COMMAND = 99H;
521  6                    CALL DISK(@CMD_BLK(INDX));
522  6                    CALL DISP_CMD_BLK;
523  6                    CMD_BLK(INDX).COMMAND = OFFH;
524  6                END;

                      /*   CASE 7 - HELP TABLE */
525  5                DO;
526  6                    CALL WRITEA(@HELP);
527  6                    CALL CO(LF);
528  6                END;

                      /*   CASE 8 - EXAMINE COMMAND BLOCK */
529  5                DO;
530  6                    CALL DISP_CMD_BLK;
531  6                END;
```

231133-44

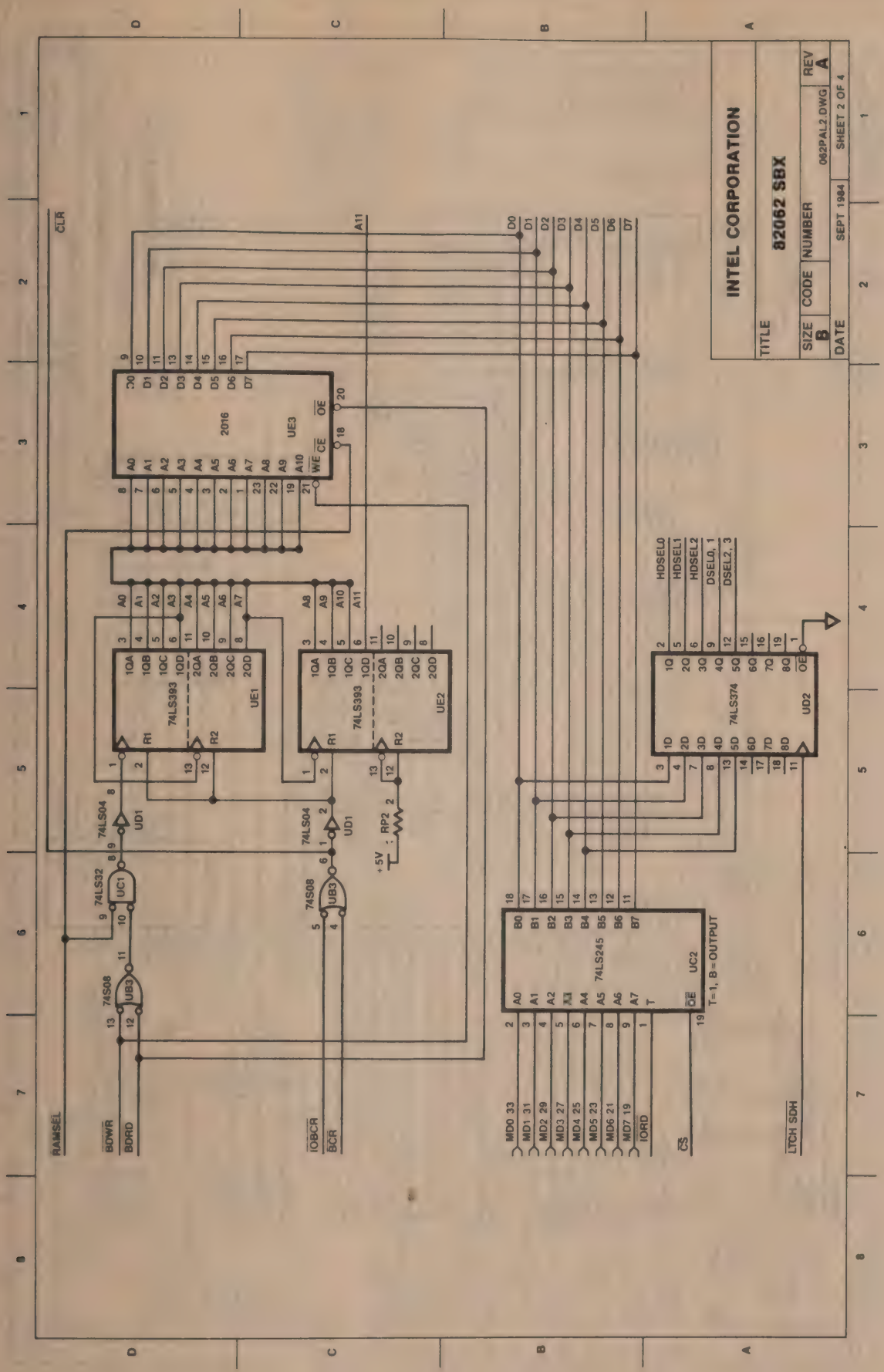
```
                                /*   CASE 9 - DISPLAY BUFFER DATA */
 532   5                        DO;
 533   6                            CALL WRITEA(@('DISPLAY ASCII<A> OR HEX<H> ? >',00));
 534   6                            CHAR = CI;
 535   6                            CALL CO(CHAR);
 536   6                            CALL CO(CR);
 537   6                            CALL CO(LF);
 538   6                            IF CHAR = 'A' THEN DO;
 540   7                                INDEX = 0;
 541   7                                DO WHILE CHAR <> ESC;
 542   8                                    DO I = 0 TO 255;
 543   9                                        CALL CO(BUFFER(INDEX + I));
 544   9                                    END;
 545   8                                    INDEX = INDEX + I;
 546   8                                    CHAR = CI;
 547   8                                END;
 548   7                            END;
 549   6                            IF CHAR = 'H' THEN DO;
 551   7                                INDEX = 0;
 552   7                                DO WHILE CHAR <> ESC;
 553   8                                    CALL DISP_HEX(@BUFFER(INDEX),256);
 554   8                                    INDEX = INDEX + 256;
 555   8                                    CHAR = CI;
 556   8                                END;
 557   7                            END;
 558   6                        END;

 559   5                    END; /* DO CASE */
 560   4                END;     /* IF */
 561   3                ELSE DO;
 562   4                    CALL WRITEA(@('*** DISK IS BUSY ***',CR,00));
 563   4                END;

 564   3                IF CMD_BLK(INDX).COMMAND = 00 THEN DO;
 566   4                    DISK_IS_NOT_BUSY = TRUE;
 567   4                    CALL DISP_STATUS;
 568   4                END;

 569   3            END;          /* FOREVER */

 570   2        END MAIN;
 571   1        END HOST_MODULE;
MODULE INFORMATION:

     CODE AREA SIZE      = 0FD8H   4056D
     CONSTANT AREA SIZE  = 093CH   2364D
     VARIABLE AREA SIZE  = 0480H   1152D
     MAXIMUM STACK SIZE  = 003EH     62D
     868 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS

DICTIONARY SUMMARY:

     31KB MEMORY AVAILABLE
     12KB MEMORY USED   (38%)
     0KB DISK SPACE USED

END OF PL/M-86 COMPILATION
```

231133-45

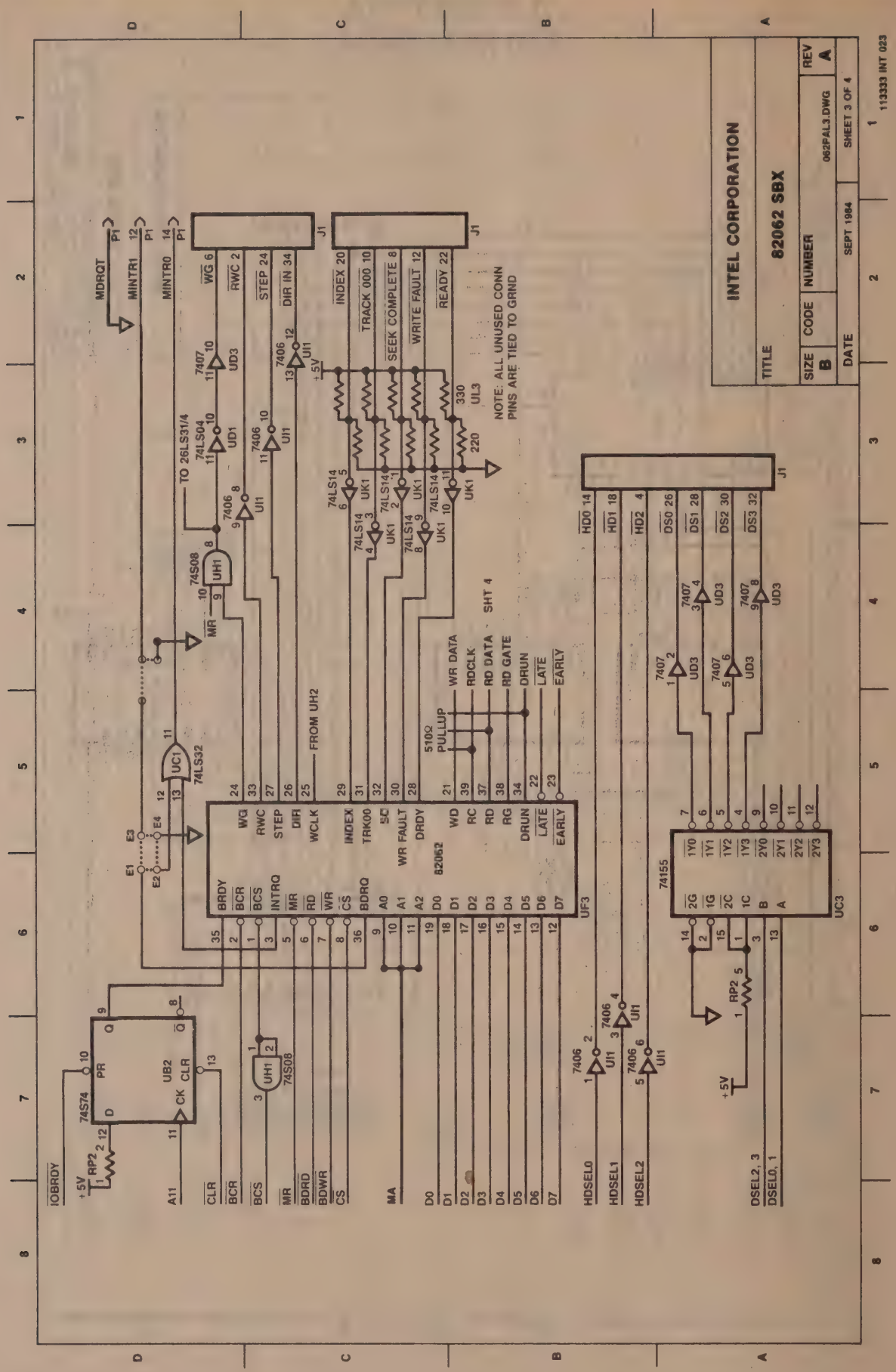
APPENDIX C
SCHEMATICS



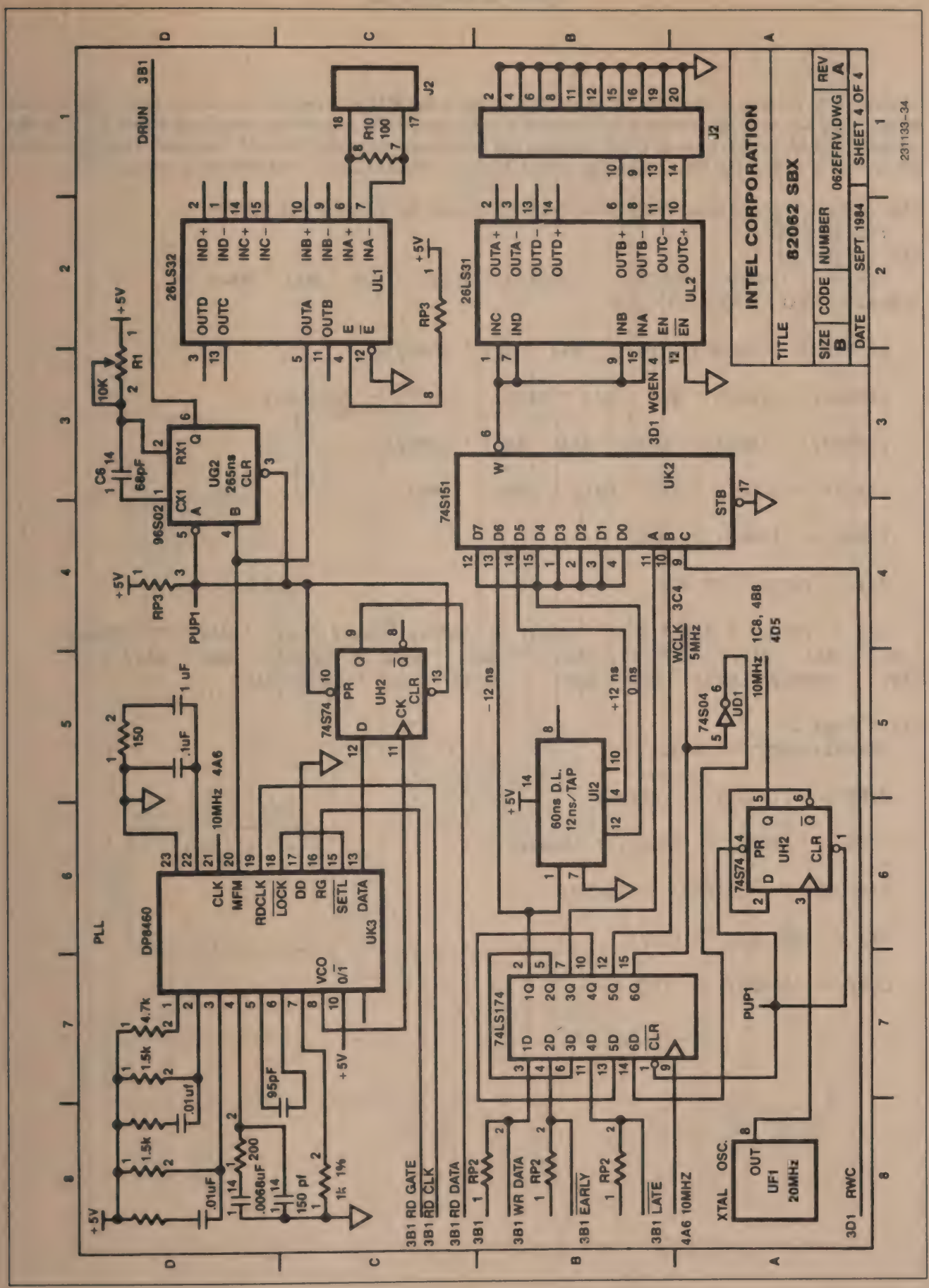
231133-002


INTEL CORPORATION
TITLE
82062 SBX
SIZE B
CODE
NUMBER
062PAL2.DWG
REV A
DATE
SEPT 1984
SHEET 2 OF 4
113332 INT 023
2016
UE3
74LS393
UE1
UE2
74LS04
UD1
74LS32
UC1
74S08
UB3
RP2
+5V
74LS374
UD2
HDSEL0
HDSEL1
HDSEL2
DSEL0, 1
DSEL2, 3
74LS245
UC2
T=1, B=OUTPUT
CLR
RAMSEL
BDWR
BDRD
IOBCR
BCR
MD0 33
MD1 31
MD2 29
MD3 27
MD4 25
MD5 23
MD6 21
MD7 19
IORD
CS
LTCH SDH

231133-002


INTEL CORPORATION
TITLE
82062 SBX
SIZE B
CODE
NUMBER
062PAL3.DWG
REV A
DATE
SEPT 1984
SHEET 3 OF 4
113333 INT 023
NOTE: ALL UNUSED CONN PINS ARE TIED TO GRND
82062
74S74
74S08
74LS04
74LS32
74LS14
74155
7407
7406
UF3
UC3
UB2
UH1
UC1
UL3
J1
RP2
+5V
510Ω PULLUP
FROM UH2
TO 26LS31/4
SHT 4
WR DATA
RDCLK
RD DATA
RD GATE
DRUN
LATE
EARLY
MDRQT
MINTR1
MINTR0
WG 6
RWC 2
STEP 24
DIR IN 34
INDEX 20
TRACK 000 10
SEEK COMPLETE 8
WRITE FAULT 12
READY 22
HD0 14
HD1 18
HD2 4
DS0 26
DS1 28
DS2 30
DS3 32
IOBRDY
A11
CLR
BCR
BCS
MR
BDRD
BDWR
CS
MA
D0
D1
D2
D3
D4
D5
D6
D7
HDSEL0
HDSEL1
HDSEL2
DSEL2, 3
DSEL0, 1

231133-002


INTEL CORPORATION
TITLE
82062 SBX
SIZE B
CODE
NUMBER
062EFRV.DWG
REV A
DATE
SEPT 1984
SHEET 4 OF 4
231133-34
DP8460
UK3
74S74
UH2
74S151
UK2
26LS32
UL1
26LS31
UL2
96S02
UG2
265ns
74S04
UD1
74LS174
60ns D.L.
12ns/TAP
UI2
J2
PLL
DRUN 3B1
WGEN 4
WCLK 3C4
5MHz
1C8, 4B8
4D5
10MHz
XTAL OSC.
UF1
20MHz
3D1 RWC
3B1 RD GATE
3B1 RD CLK
3B1 RD DATA
3B1 WR DATA
3B1 EARLY
3B1 LATE
4A6 10MHZ
PUP1
RP3
RP2
R1
R10
C6
68pF
10K
+5V
-12 ns
+12 ns
0 ns

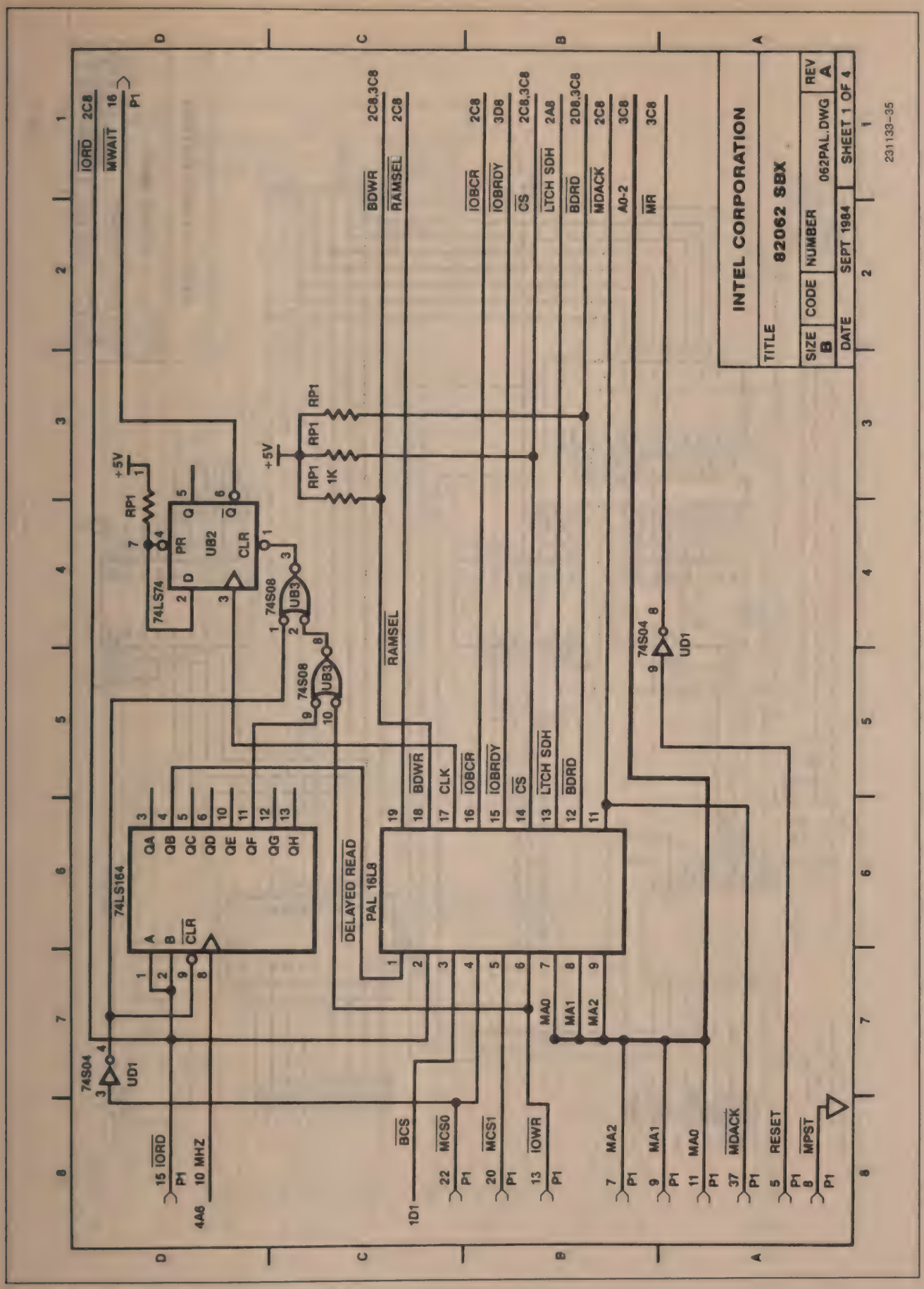
231133-002

APPENDIX D

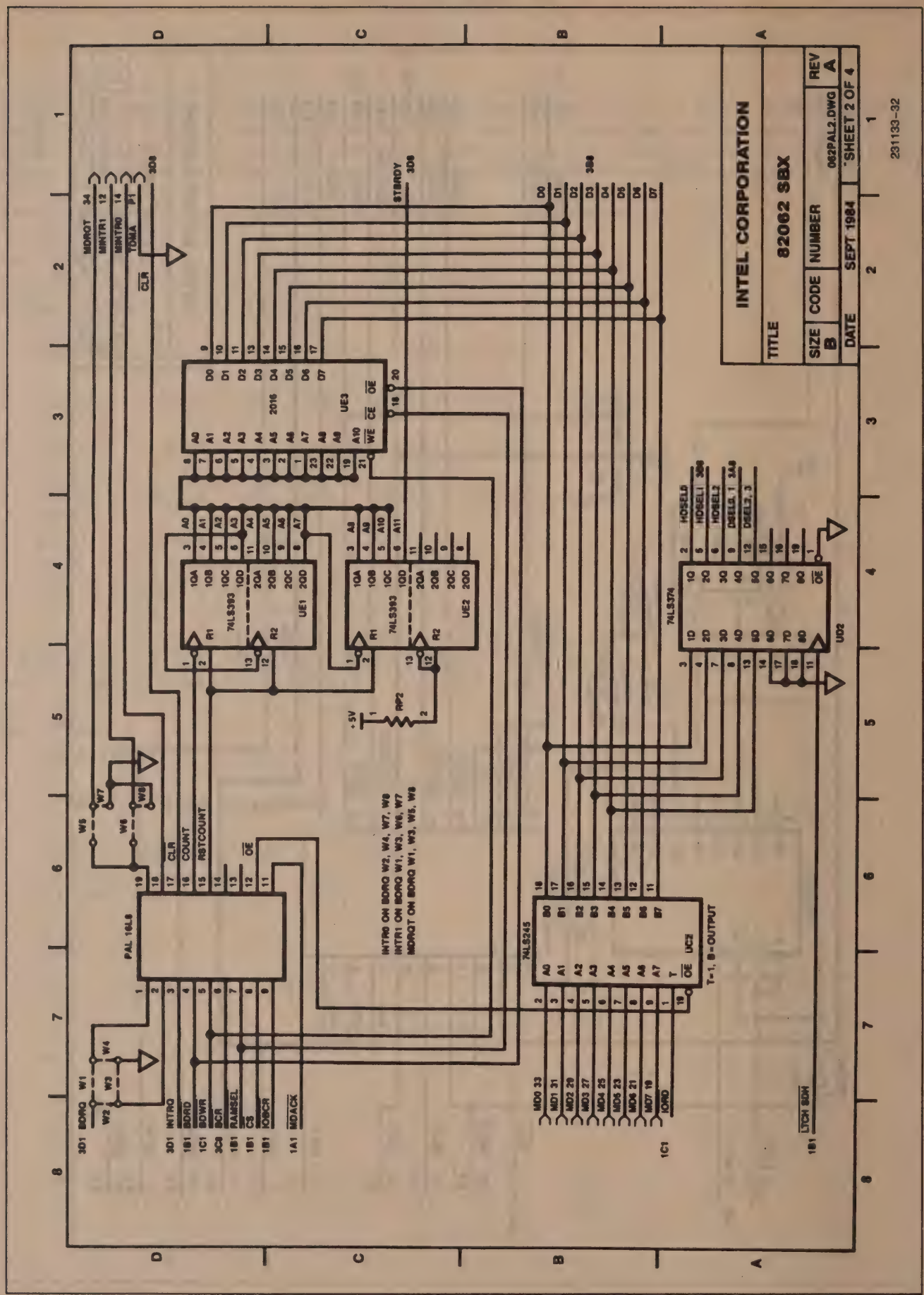
This appendix contains a schematic of the previous design using PAL's to replace the random logic. The previous design could not do DMA transfers and inserted a large delay when transferring data from buffer RAM to the system. The PAL version does do DMA transfers and buffer reads happen at full SBX bus speed. One other minor change was to replace the 500 ns delay line with a 74LS164, which is a more cost effective solution.

This schematic is only a paper design since only random logic was replaced with the PAL's.

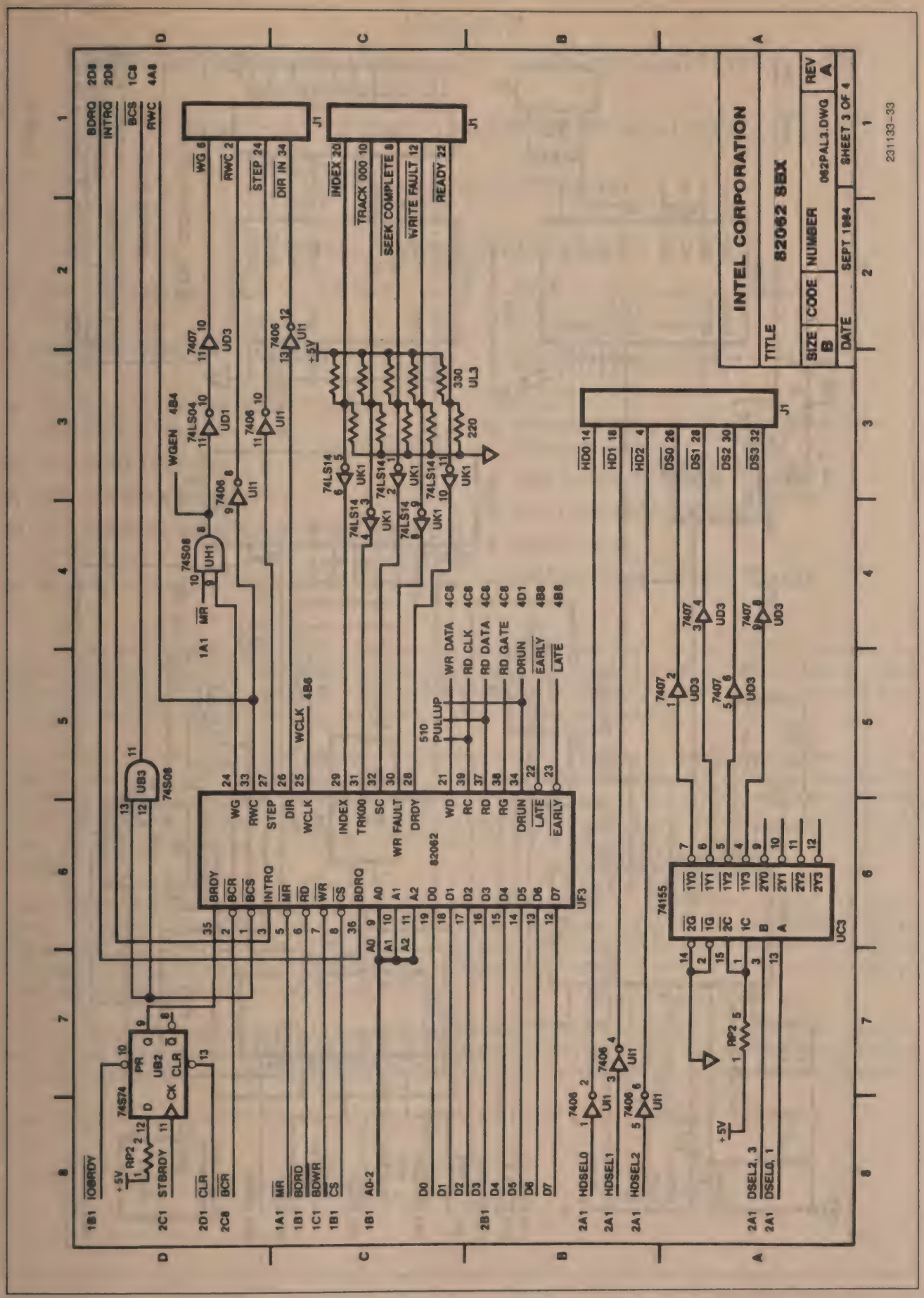
```
   PAL Equation's
PAL - Page 1:
   BDRD/ = (IORD/ * MDACK/) + (IORD/ * MCS0/ * MA0 * MA1 * MA2) +
(DELAYED-READ/ * CLK) IF BCS

   LTCHSDH/ = (MCS0/ * MA0/ * MA1 * MA2 * IOWR/)

   RAMSEL/ = (MCS0 * MA0 * MA1 * MA2) + (BCS/) + (MDACK/)

   IOBRDY/ = (MCS1/ * MA0/ * MA1 * MA2/ * IOWR/)

   IOBCR/ = (MCS1/ * MA0 * MA1/ * MA2/ * IOWR/)

   BDWR/ = (IOWR/) IF BCS

   CS/ = (MCS0/) IF BCS

   CLK = (MCS0/ * MA0 * MA1/ * MA2/) + (MCS0/ * MA0/ * MA1 * MA2/) + (MCS0/
* MA0 * MA1 * MA2/) + (MCS0/ * MA0/ * MA1/ * MA2) + (MCS0/ * MA0 * MA1/ *
MA2) + (MCS0/ * MA0/ * MA1 * MA2) + (MCS0/ * MA0 * MA1 * MA2)

PAL - Page 2:
   MINTR1/MDRQT = (PIN1)

   MINTR0 = (PIN2) + (INTRQ)

   COUNT = (BDWR/ + BDRD/) * (RAMSEL/)

   RSTCOUNT = (IOBCR/) + (BCR/)

   OE/ = (MDACK/) + (CS/)

   CLR/ = (IOBCR/) + (BCR/)
```


INTEL CORPORATION
TITLE
82062 SBX
SIZE B
CODE
NUMBER
062PAL.DWG
REV A
DATE
SEPT 1984
SHEET 1 OF 4
231133-35
74LS74
UB2
74S08
UB3
74LS164
PAL 16L8
74S04
UD1
RP1
1K
+5V
DELAYED READ
RAMSEL
BDWR
CLK
IOBCR
IOBRDY
CS
LTCH SDH
BDRD
MDACK
A0-2
MR
MA0
MA1
MA2
BCS
MCS0
MCS1
IOWR
RESET
MPST
IORD
MWAIT
10 MHZ

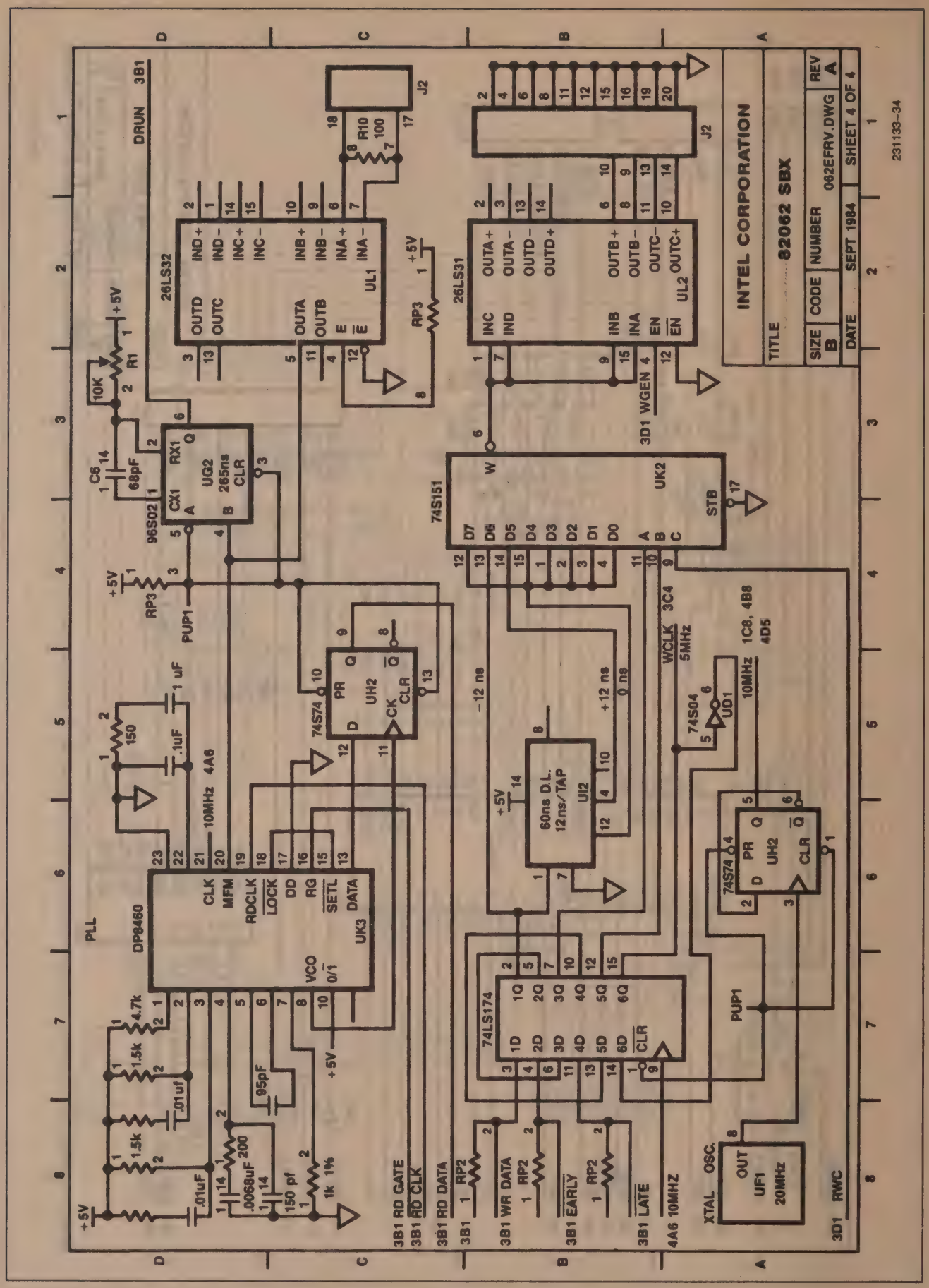
231133-002


INTEL CORPORATION
TITLE
82062 SBX
SIZE B
CODE
NUMBER
062PAL2.DWG
REV A
DATE
SEPT 1984
SHEET 2 OF 4
231133-32
UE3
2016
74LS393
UE1
UE2
74LS374
UD2
74LS245
UC2
T-1, B-OUTPUT
PAL 16L8
RP2
+5V
INTR0 ON BDRQ W2, W4, W7, W8
INTR1 ON BDRQ W1, W3, W6, W7
MDROT ON BDRQ W1, W3, W5, W8
MDROT
MINTR1
MINTR0
TOMA
STBRDY
HOSEL0
HOSEL1
HOSEL2
DSEL0, 1
DSEL2, 3
INTRO
BDRQ
BDWR
BCR
RAMSEL
CS
IOBCR
MDACK
MD0 33
MD1 31
MD2 29
MD3 27
MD4 25
MD5 23
MD6 21
MD7 19
IORD
LTCH BDH

231133-002


INTEL CORPORATION
TITLE
82062 SBX
SIZE B
CODE
NUMBER
062PAL3.DWG
REV A
DATE
SEPT 1984
SHEET 3 OF 4
231133-33
82062
74S08
74S04
74S74
74LS14
7407
7406
74155
WR DATA
RD CLK
RD DATA
RD GATE
DRUN
EARLY
LATE
PULLUP
WGEN
WCLK
+5V

231133-002


INTEL CORPORATION
TITLE 82062 SBX
SIZE B
CODE
NUMBER 062EFRV.DWG
REV A
DATE SEPT 1984
SHEET 4 OF 4
231133-34
26LS32
26LS31
74S151
74S74
74LS174
DP8460
96S02
60ns D.L. 12ns/TAP
UL1
UL2
UK2
UK3
UG2
UH2
UI2
UF1
20MHz
J2
DRUN 3B1
PLL
PUP1
WCLK 3C4
5MHz
10MHz 1C8, 4B8 4D5
3D1 WGEN 4
3B1 RD GATE
3B1 RD CLK
3B1 RD DATA
3B1 WR DATA
3B1 EARLY
3B1 LATE
4A6 10MHZ
XTAL OSC.
3D1 RWC

231133-002

UPI-452
CHMOS SINGLE CHIP SLAVE MICROCONTROLLER

83452 - 8K × 8 Mask Programmable Internal ROM

87452 - 8K × Internal EPROM

80452 - External ROM/EPROM

- 83452/87452/80452:1.6 to 16 MHz Clock Rate
- Software Compatible with the MCS-51 Family
- 128-Byte Bi-Directional FIFO Slave Interface
- Two DMA Channels
- 256 × 8-Bit Internal RAM
- 34 Additional Special Function Registers
- 40 Programmable I/O Lines
- Two 16-Bit Timer/Counters
- Boolean Processor
- Bit Addressable RAM
- 8 Interrupt Sources
- Programmable Full Duplex Serial Channel
- 64K Program Memory Space
- 64K Data Memory Space
- 68-Pin PGA
  (See Packaging Spec., Order: #231369)

The Intel 83452/87452/80452 Universal Peripheral Interface, UPI™-452, is a 68 pin CHMOS Microcontroller with a sophisticated bi-directional FIFO buffer interface on the slave bus and a two channel DMA processor on-chip. The UPI-452 is the newest member of Intel's UPI family of products. It is a general-purpose slave microcontroller that allows the designer to grow a customized interface solution.

The UPI-452 contains a complete 80C51 CPU core with twice the on-chip data and program memory. The sophisticated slave FIFO module acts as a buffer between the UPI-452 internal CPU and the external host CPU. To both the external host and the internal CPU, the FIFO module looks like a bi-directional bottomless buffer that can both read and write data. The FIFO manages the transfer of data independent of the UPI-452 core CPU and generates an interrupt or DMA request to either CPU, host or internal, as a FIFO service request.

The FIFO consists of two channels:the Input FIFO and the Output FIFO. The division of the FIFO module array, 128 bytes, between Input channel and Output channel is programmable by the user. Each FIFO byte has an additional ninth bit to distinguish between a data byte and a Data Stream Command byte. Additionally, Immediate Commands allow direct, interrupt drive, bi-directional communication between the UPI-452 internal CPU and external host CPU, bypassing the FIFO.

The on-chip DMA processor allows high speed data transfers from one writeable memory space to another. As many as 64K bytes can be transferred in a single DMA operation. Three distinct memory spaces may be used in DMA operations; Internal Data Memory, External Data Memory, and the Special Function Registers.

Like the 80C51, the CHMOS UPI-452 has Normal, Idle and Power Down Modes.

November 1985
Order Number: 231428-001

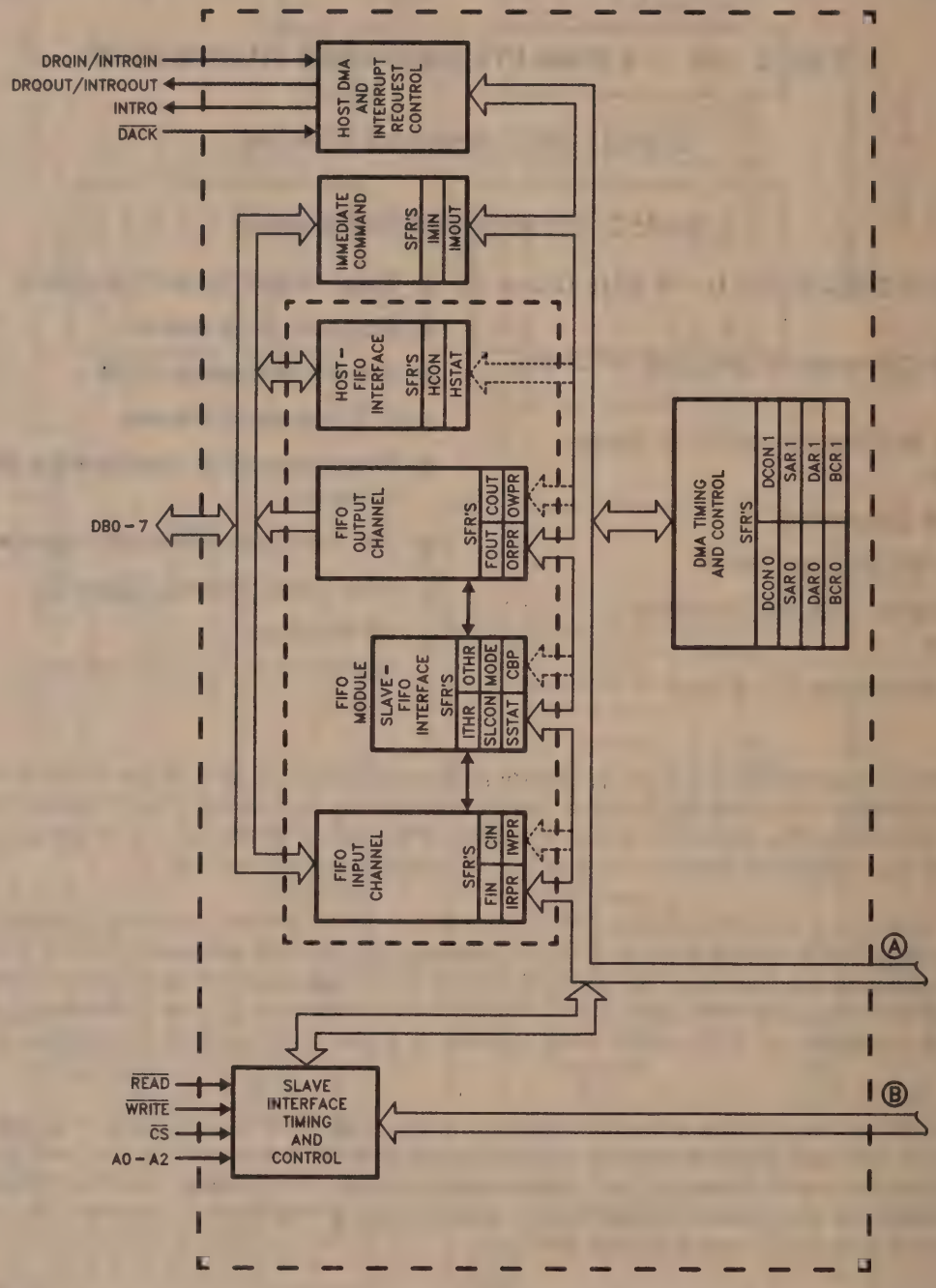


Figure 1. Architectural Block Diagram

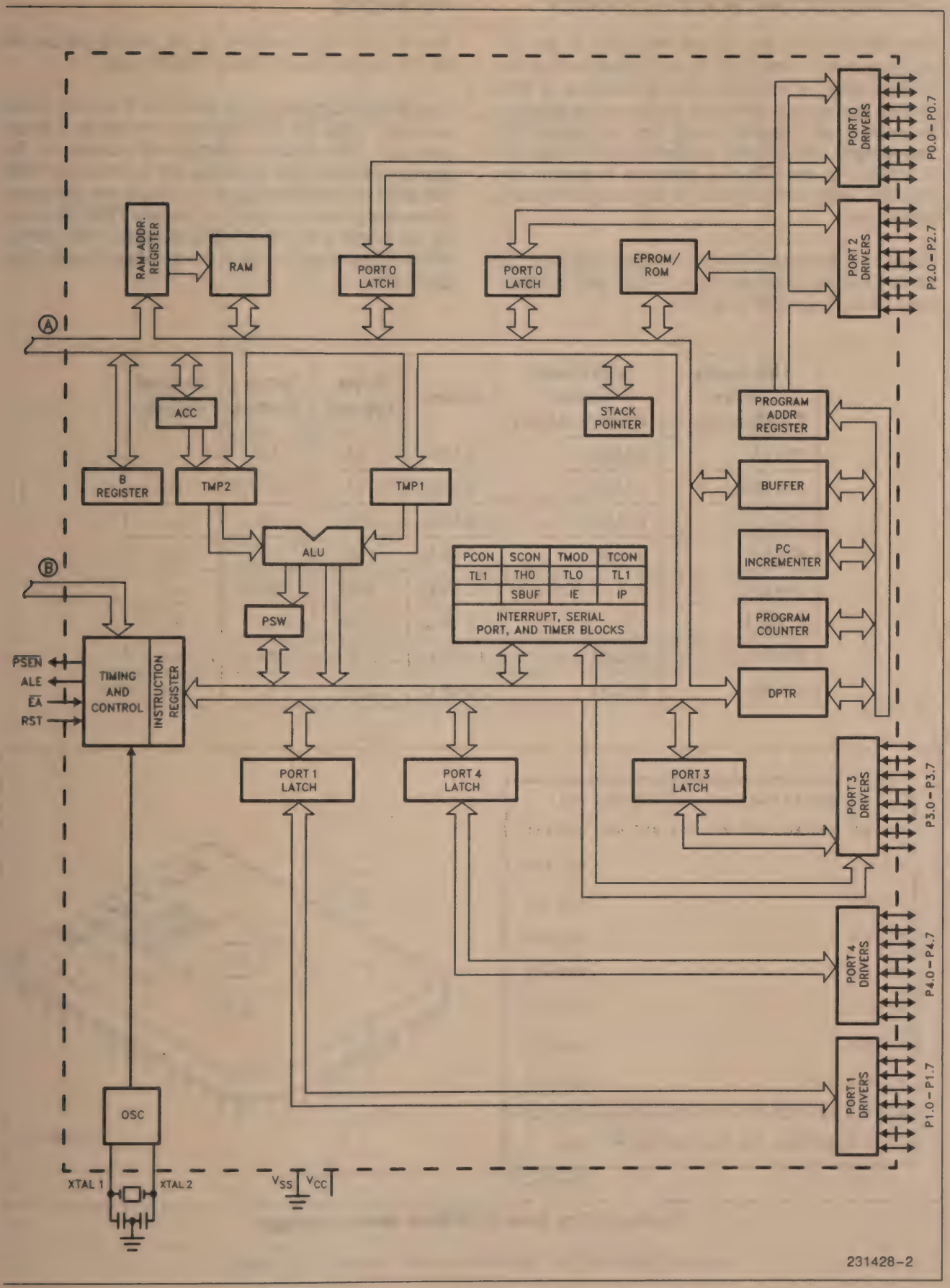


Figure 1. Architectural Block Diagram (Continued)

UPI MICROCONTROLLER FAMILY

The UPI-452 joins the current members of the UPI microcontroller family. UPI's are derivatives of the MCS™ family of microcontrollers. Because of their on-chip system bus interface, UPI's are designed to be system bus "slaves", while their microcontroller counterparts are intended as system bus "masters". In addition to the UPI-452, which is based on the MCS-51 family, Intel makes the following UPI microcontrollers:

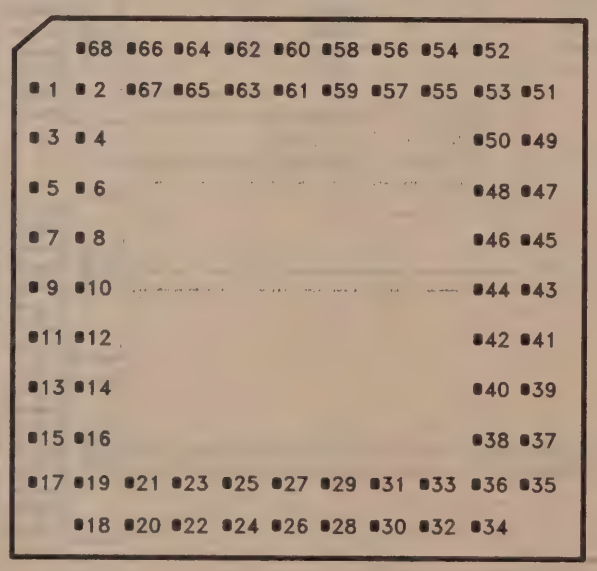
These UPI Microcontrollers are fully supported by Intel's EPROM programmers (iUP-201) and development tools (ICE, ASM and PLM).

Packaging

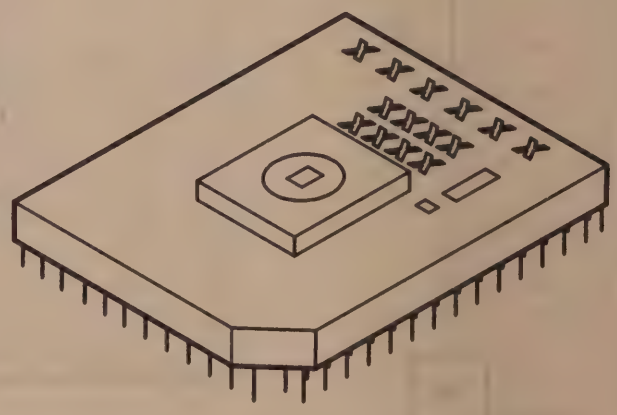
The first UPI-452 versions to be offered will be the 87452 (EPROM) and 83452 (ROM-Less).

The 80452 comes in a 68-pin PGA (Pin Grid Array) package, while the 87452 will be offered in a hybrid package. This hybrid package will consist of the standard 68-pin PGA package with a 2764A EPROM soldered on top (see Figure 2). These two packages allow designers to use either on-chip EPROM or external memory for their initial designs. The 83452 (ROM version) will come in the standard 68-pin PGA package.

UPI Family (Slave Configuration)	MCS Family (Master Configuration)	Speed	RAM (Bytes)	ROM (Bytes)	EPROM (Bytes)
8041A	8048AH	6 MHz	64	1K	—
8741AH	8748H	6 MHz	64	—	1K
8042	8049AH	12 MHz	128	2K	—
8742H	8749H	12 MHz	128	—	2K
8042A	—	12 MHz	256	2K	—
8742AH	—	12 MHz	256	—	2K
83452	80C51	16 MHz	256	8K	—
87452	80C51	16 MHz	256	—	8K

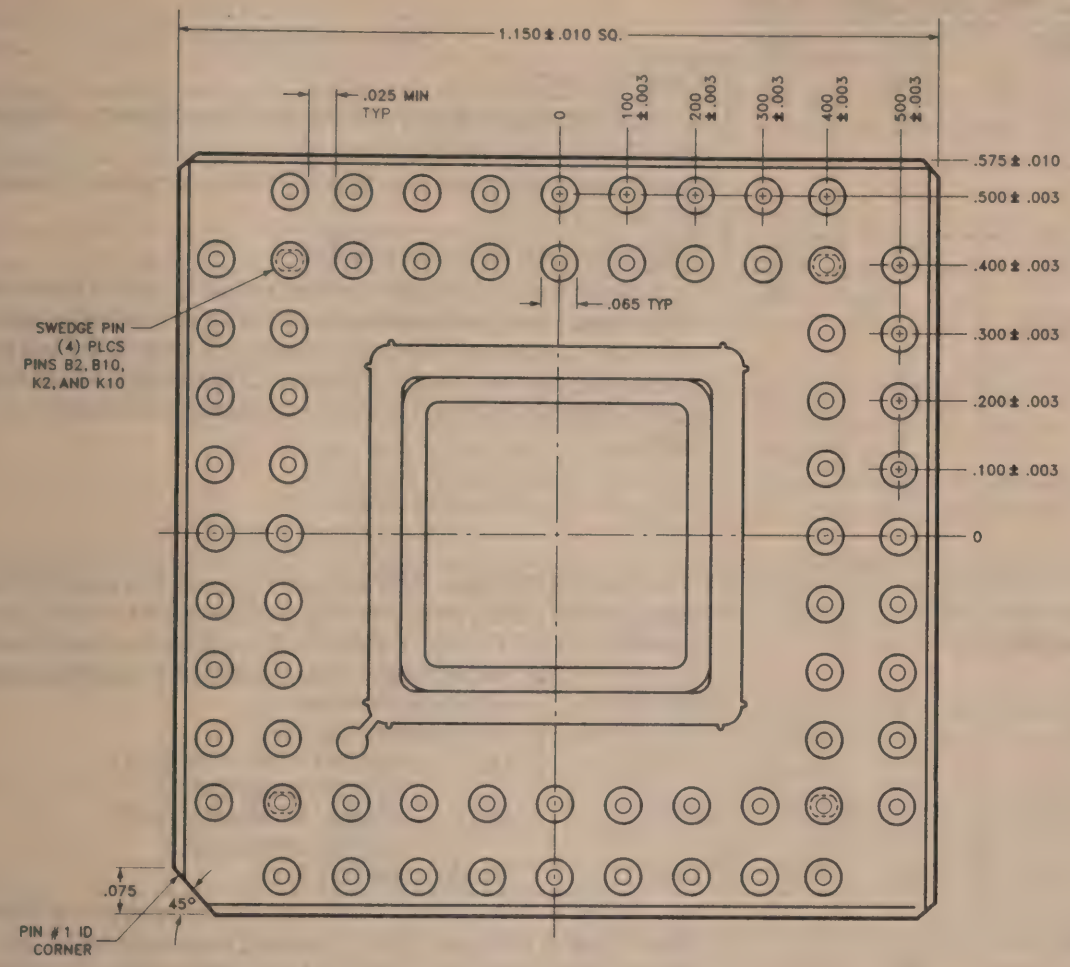


231428-3

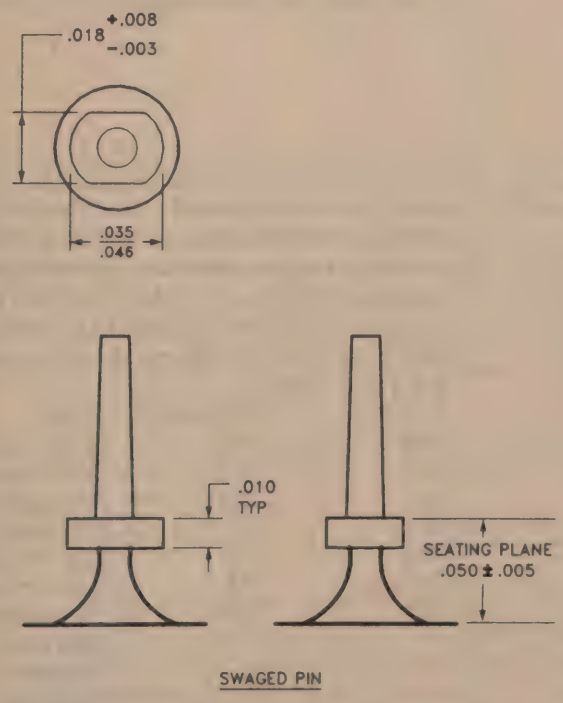


231428-4

Figure 2a. Top View of UPI-452 68-Pin Package



231428-5



231428-6

Figure 2b. Bottom View of UPI-452 68-Pin PGA Package

UPI-452 PIN DESCRIPTIONS

Symbol	Pin #	Type	Name and Function
V_{SS}	9/43	I	Circuit Ground.
V_{CC}	60	I	+5V power supply during normal, idle, power down, programming and verification operation.
XTAL1	38	I	Input to the oscillator's high gain amplifier. A crystal or external source can be used.
XTAL2	39	O	Output from the high gain amplifier.
Port 0 (AD0–AD7) PO.0 .1 .2 .3 .4 .5 .6 PO.7	 8 10 11 12 13 14 15 16	I/O	Port 0 is an 8-bit open drain bi-directional I/O port. It is also the multiplexed low-order address and data local expansion bus during accesses to external memory. It is used for data input and output during programming and verification. External pullups are required during program verification. Port 0 can sink/source eight LS TTL inputs.
Port 1 (A0–A7) (HLD, HLDA) P1.0 .1 .2 .3 .4 .5 .6 P1.7	 7 6 5 4 3 2 1 68	I/O	Port 1 is an 8-bit quasi-bi-directional I/O port. It is used for low-order address byte during programming and verification. Port 1 can sink/source four LS TTL inputs. Pins P1.5 and P1.6 are multiplexed with HLD and HLDA respectively whose functions are defined as below: Port Pin — Alternate Function P1.0–P1.4 — (No Special Function) P1.5 — HLD —Parallel interface's hold input/output signal P1.6 — HLDA —Parallel interface's hold acknowledge output P1.7 — (No Special Function)
Port 2 (A8–A15) P2.0 .1 .2 .3 .4 .5 .6 .7	 29 28 27 25 24 23 22 21	I/O	Port 2 is an 8-bit quasi-bi-directional I/O port. It also emits the high-order 8 bits of address when accessing local expansion bus external memory (or during 87452 programming and verification). Port 2 can sink/source four LS TTL inputs.
Port 3 P3.0 .1 .2 .3 .4 .5 .6 P3.7	 67 66 65 64 63 62 61 59	I/O	Port 3 is an 8-bit quasi-bi-directional I/O port. It is also multiplexed with the interrupt, timer, local serial channel, RD/ and WR/ functions that are used by various options. The output latch corresponding to a Special Function Register must be programmed to a one (1) for that function to operate. Port 3 can sink/source four LS TTL inputs. The alternate functions assigned to the pins of Port 3 are as follows: Port Pin — Alternate Function P3.0 — RxD — Serial input port P3.1 — TxD — Serial output port P3.2 — INT0 — Interrupt 0 Input P3.3 — INT1 — Interrupt 1 Input P3.4 — T0 — Input to counter 0 P3.5 — T1 — Input to counter 1 P3.6 — WR/ — The write control signal latches the data from Port 0 outputs into the External Data Memory on the local bus. P3.7 — RD/ — The read control signal latches the data from Port 0 outputs on the local bus.

UPI-452 PIN DESCRIPTIONS (Continued)

Symbol	Pin #	Type	Name and Function
Port 4 P4.0 .1 .2 .3 .4 .5 .6 .7	I/O 30 31 32 33 34 35 36 37		Port 4 is an 8-bit quasi-bi-directional I/O port. Port 4 can sink/source four TTL inputs. It is also used as the control signals during EPROM programming and verification as follows: Port Pin — Alternate Function P4.5 — '1' during program and verify P4.6 — '0' during program and verify P4.7 — '0' during verify - used as output enable '1' during programming w/ ALE = 0 Note: see Programming and Verification Characteristics in AC/DC Specification section.
RST	20	I	A high level on this pin for two machine cycles while the oscillator is running resets the device. An internal pulldown resistor permits Power-on reset using only a capacitor connected to V_{CC}. This pin does not receive the power down voltage as is the case for HMOS MCS-51 family members. This function has been transferred to the V_{CC} pin.
ALE/PGM	18	I/O	Provides Address Latch Enable output used for latching the address into external memory during normal operation. Receives the program pulse input during EPROM programming. ALE can sink/source eight LS TTL inputs.
PSEN	19	O	The Program Store Enable output is a control signal that enables the external Program Memory to the bus during normal fetch operation. PSEN can sink/source eight LS TTL inputs.
EA	17	I	When held at TTL high level, the UPI-452 executes instructions from the internal ROM/EPROM when the PC is less than 8192 (8K, 200H). When held at a TTL low level, the UPI-452 fetches all instructions from external Program Memory.
DBO DB1 DB2 DB3 DB4 DB5 DB6 DB7	58 57 56 55 54 53 52 51	I/O	Slave Data Bus is an 8-bit bi-directional bus. It is used to transfer data and commands between the UPI-452 and the host processor. This bus can sink/source eight LS TTL inputs.
$\overline{CS}$	44	I	This pin is the Chip Select of the UPI-452.
A0 A1 A2	40 41 42		These three address lines are used to interface with the host system. They define the UPI-452 operations. The interface is compatible with the Intel microprocessors and the MULTIBUS.
$\overline{READ}$	46	I	This pin is the read strobe from the host CPU. Activating this pin causes the UPI-452 to place the contents of the Output FIFO (either a command or data) or the Host Status/Control Special Function Register on the Slave Data Bus.
$\overline{WRITE}$	47	I	This pin is the write strobe from the host. Activating this pin will cause the value on the Slave Data Bus to be written to the Input FIFO as a command or data.
$\overline{DRQIN}$/ INTRQIN	49	O	This pin requests an input transfer whenever the Input Channel requires data.
DRQOUT/ INTRQOUT	48	O	This output pin requests an output transfer whenever the Output Channel requires service. If the external host to UPI-452 DMA is enabled, and a Data Stream Command is at the Output FIFO, DRQOUT is deactivated and INTRQ is activated (see 'GENERAL PURPOSE DMA CHANNELS' section).

UPI-452 PIN DESCRIPTIONS (Continued)

Symbol	Pin #	Type	Name and Function
INTRQ	50	O	This output pin is used to interrupt the host processor when an Immediate Command Out or an error condition is encountered. It is also used to interrupt the host processor when the FIFO requests service if the DMA is disabled and INTRQIN and INTRQOUT are not used.
$\overline{DACK}$	45	I	This pin is the DMA acknowledge for the Slave Data Bus Input and Output Channels. When activated, a write command will cause the data on the Slave Data Bus to be written as data to the Input Channel (to the Input FIFO). A read command will cause the Output Channel to output data (from the Output FIFO) on to the Slave Data Bus. This pin should be driven high (+5V) in systems which do not have a DMA controller (see Address Decoding).
V_{CC}/V_{PP}	26	I	+5V power supply during operation. The V_{CC} pin receives the +12V EPROM programming and verification supply voltage. It is also the standby power pin for power down mode.

ARCHITECTURAL OVERVIEW

Introduction

The UPI-452 slave microcontroller is essentially an 80C51 with double the program and data memory, a slave interface which allows it to be connected directly to the host system bus as a peripheral, a FIFO buffer module, a two channel DMA processor, and a fifth I/O port (Figure 3). The UPI-452 retains all of the 80C51 architecture, and is fully compatible with the MCS-51 instruction set.

The Special Function Register (SFR) interface concept introduced in the MCS-51 family of microcontrollers has been expanded in the UPI-452. To the 25 Special Function Registers of the MCS-51, the UPI-452 adds 34 more. These additional Special Function Registers, like those of the MCS-51, provide access to the UPI-452 functional elements including the FIFO, DMA and added interrupt capabilities. Several of the 80C51 core Special Function Registers have also been expanded to support added features of the UPI-452.

This data sheet describes the unique features of the UPI-452. Refer to the 80C51 data sheet for a description of the UPI-452's core CPU functional blocks including;

— Timers/Counters
— I/O Ports
— Interrupt timing and control (other than FIFO and DMAinterrupts)
— Serial Channel
— Local Expansion Bus
— Program/Data Memory structure
— Power-Saving Modes of Operation *
— CHMOS Features
— Instruction Set

* except 87452 hybrid package

Figure 3 contains a conceptual block diagram of the UPI-452. Figure 4 provides a functional block diagram.

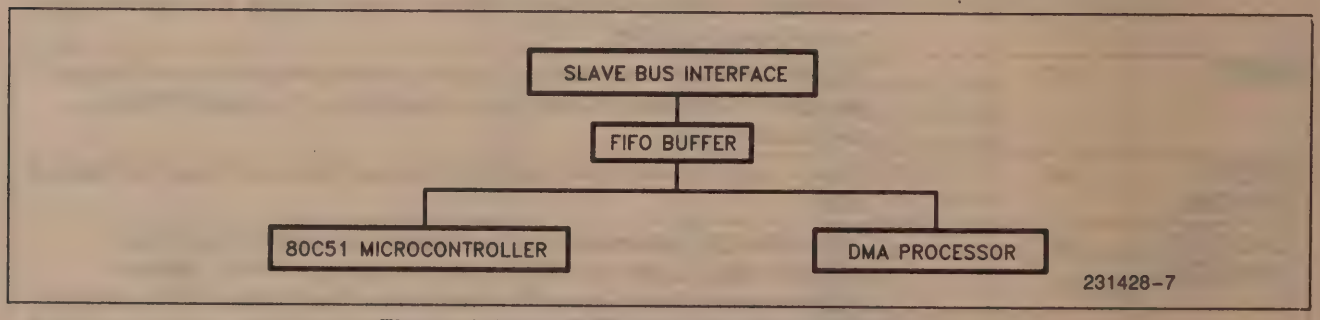


Figure 3. UPI-452 Conceptual Block Diagram

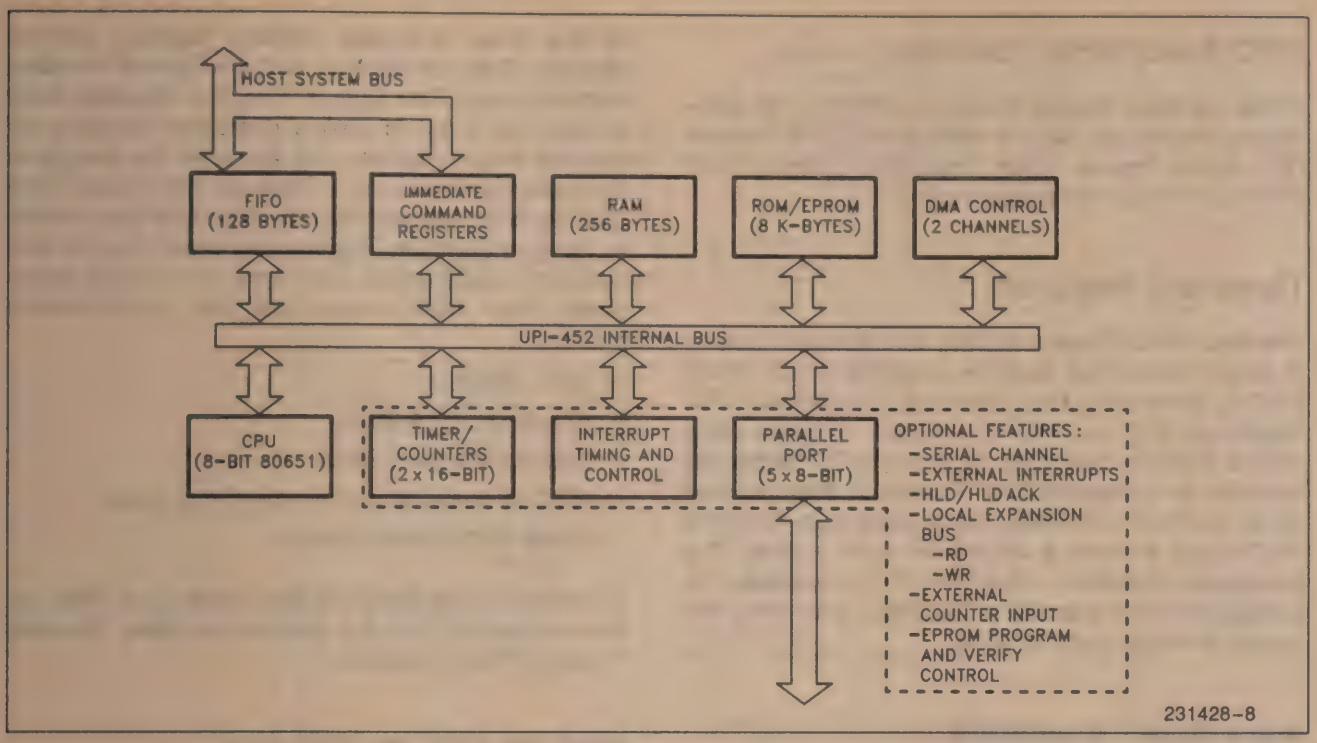


Figure 4. UPI-452 Functional Block Diagram

FIFO Buffer Interface

A unique feature of the UPI-452 is the incorporation of a 128 byte FIFO array at the host-slave interface. The FIFO allows asynchronous bi-directional transfers between the host CPU and the internal CPU. The division of the 128 bytes between Input and Output channels is user programmable allowing maximum flexibility. If the entire 128 byte FIFO is allocated to the Input channel, a high performance Host can transfer up to 128 bytes at one time, then dedicate its resources to other functions while the internal CPU processes the data in the FIFO. Various handshake signals allow the external Host to operate independently and without frequent monitoring of the UPI-452 internal CPU. The FIFO Buffer insures that the slave processor receives data in the same order that it was sent by the host without the need to keep track of addresses. Three slave bus interface handshake methods are supported by the UPI-452: DMA, Interrupt and Polled.

The FIFO is nine bits wide. The ninth bit acts as a command/data flag. Commands written to the FIFO by either the host or internal CPU are called Data Stream Commands or DSCs. DSCs are written to the input FIFO by the Host via a unique external address. DSCs are written to the output FIFO by the internal CPU via the COMMAND OUT Special Function Register (SFR). When encountered by the host or internal CPU a Data Stream Command can be used as an address vector to user defined service routines. DSCs provide synchronization of data and commands between the Host and internal CPU.

FIFO PROGRAMMABLE FEATURES

Size of Input/Output Channels

The 128 bytes of FIFO space can be allocated between the Input and Output channels via the Channel Boundary Pointer (CBP) SFR. This register contains the number of address locations assigned to the Input channel. The remaining address locations are automatically assigned to the Output FIFO. The CBP SFR can only be programmed by the internal CPU during Freeze Mode (See FIFO-External Host Interface Freeze Mode description). The CBP is initialized to 40H (64 bytes) upon reset, and can range from OOH–7FH.

The number in the Channel Boundary Pointer SFR is actually the first address location of the Output FIFO. Writing to the CBP SFR reassigns the Input and Output FIFO address space. Whenever the CBP is written, the Input FIFO pointers are reset to zero and the Output FIFO pointers are set to the value in the CBP SFR.

All of the FIFO space may be assigned to one channel. In such a situation the other channel's data path consists of a single SFR (FIFO IN/COMMAND IN or FIFO OUT/COMMAND OUT SFR) location.

FIFO Read/Write Pointers

These normally operate in auto-increment (and auto-rollover) mode, but can be reassigned by the internal CPU during Freeze Mode (See FIFO-External Host Interface Freeze Mode description).

Threshold Register

The input FIFO Threshold SFR contains the number of empty bytes that must be available in the Input FIFO to generate a Host interrupt. The Output FIFO Threshold SFR contains the number of bytes, data and/or DSC(s), that must be in the FIFO before an interrupt is generated.The Threshold feature prevents the Host from being interrupted each time the FIFO needs to load or unload one byte of data. The thresholds, therefore, allow the FIFO's operation to be adjusted to the speed of the Host, optimizing the overall interface performance.

Immediate Commands

The UPI-452 provides, in addition to data and DSCs, a third direct means of communication between the external host and internal CPUs called Immediate Commands. As the name implies, an Immediate Command is available to the receiving CPU immediately, via an interrupt, without being entered into the FIFO as are Data Stream Commands. Like Data Stream Commands, Immediate Commands are written either via a unique external address by the host CPU, or via dedicated SFR by the internal CPU to the external host CPU.

The DSC and/or Immediate Command interface may be defined as either Interrupt or Polled under user program control via the Interrupt Enable (IE) and Interrupt Enable Priority (IEP) Special Function Registers, for the internal CPU and via the Host Control SFR for the external host CPU.

DMA

The UPI-452 contains a two channel internal DMA controller which allows transfer of data between any of the three writeable memory spaces: Internal Memory, External Memory and the Special Function Register array. The Special Function Register array appears as a set of unique dedicated memory addresses which may be used as either the source or destination address of a DMA transfer. Each DMA channel is independently programmable via dedicated Special Function Registers for mode, source and destination addresses, and byte count to be transferred. Each DMA channel has five programmable modes:

— Burst Mode

— Alternate Cycle Mode

— External Demand Mode

— FIFO Demand and Alternate Cycle Mode

— Serial Port Demand Mode

A complete description of each mode and DMA operation may be found in the section titled "General Purpose DMA Channels".

FIFO/SLAVE INTERFACE FUNCTIONAL DESCRIPTION

Overview

The FIFO is a 128 Byte RAM array with recirculating pointers to manage the read and write accesses. The FIFO consists of an Input and an Output channel. Access cycles to the FIFO by the internal CPU and external Host are interleaved and appear to be occurring concurrently to both the internal CPU and external Host. Interleaving access cycles ensures efficient use of this shared resource. The internal CPU accesses the FIFO in the same way it would access any of the Special Function Registers e.g., direct and register indirect addressing as well as arithmetric and logical instructions.

Input FIFO Channel

The Input FIFO Channel provides for data transfer from the external Host to the internal CPU (Figure 5). The registers associated with the Input Channel during normal operation are listed in Table 1*.

Table 1. Input FIFO Channel Registers

	Register Name	Description
1)	Input Buffer Latch	Host CPU Write only
2)	FIFO IN SFR	Internal CPU Read only
3)	COMMAND IN SFR	Internal CPU Read only
4)	Input FIFO Read Pointer SFR	Internal CPU Read only
5)	Input FIFO Write Pointer SFR	Internal CPU Read only
6)	Input FIFO Threshold SFR	Internal CPU Read only

*See "FIFO-EXTERNAL HOST INTERFACE FREEZE MODE" section for Freeze Mode SFR characteristics description.

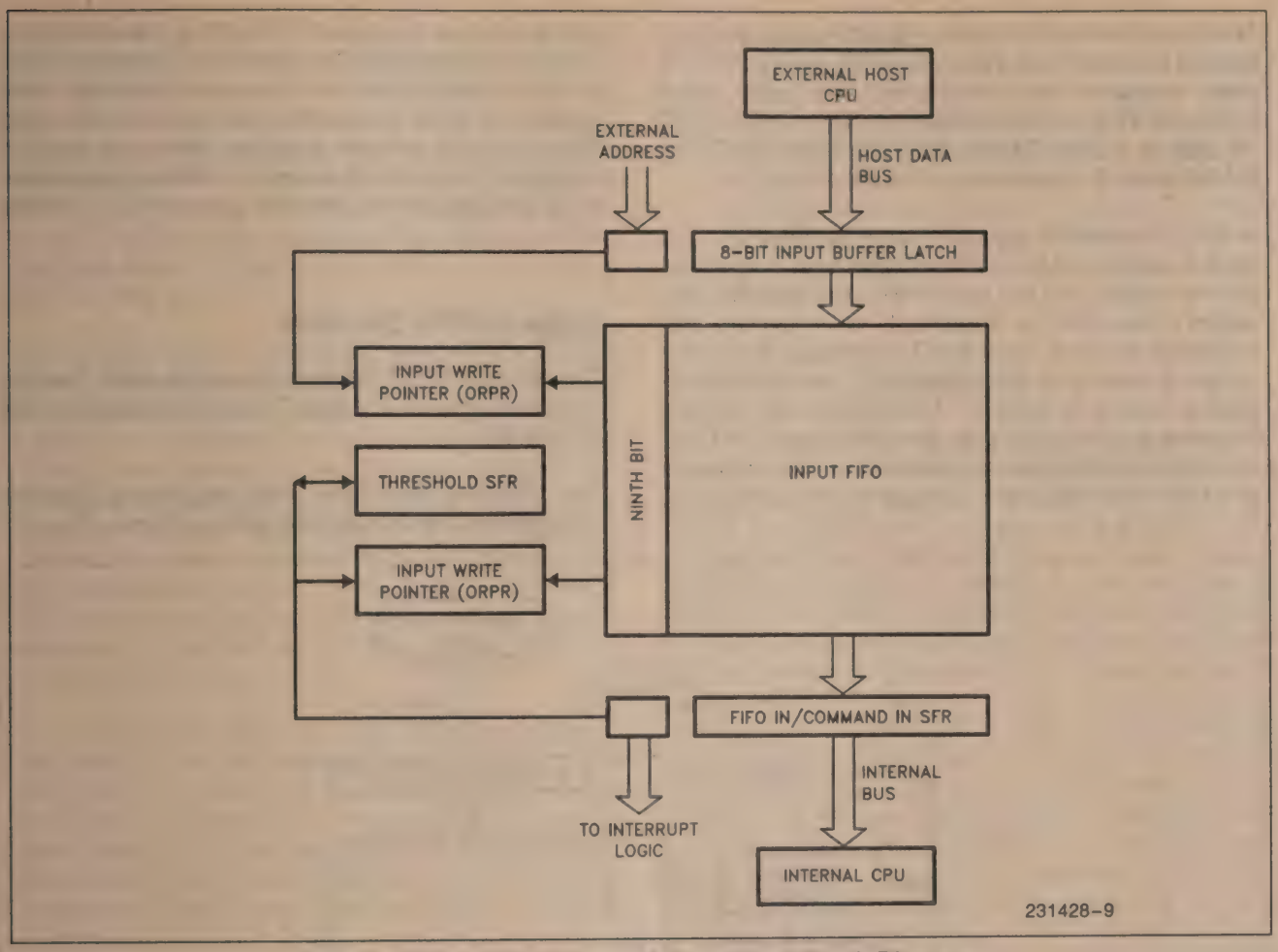


Figure 5. Input FIFO Channel Functional Block Diagram

The host CPU writes data and Data Stream Commands into the Input Buffer Latch on the rising edge of the external WR signal. External addressing determines whether the byte is a data byte or Data Stream Command and the FIFO logic sets the ninth bit of the FIFO accordingly as the byte is moved from the Input Buffer Latch into the FIFO. A "1" in the ninth bit indicates that the incoming byte is a Data Stream Command. The internal CPU reads data bytes via the FIFO IN SFR, and Data Stream Commands via the COMMAND IN SFR.

A Data Stream Command will generate an interrupt to the internal CPU prior to being read and after completion of the previous operation. The DSC can then be read via the COMMAND IN SFR. Data can only be read via the FIFO IN SFR and Data Stream Commands via the COMMAND IN SFR. Attempting to read Data Stream Commands as data by addressing the FIFO IN SFR will result in "0FFH" being read, and the Input FIFO Read Pointer will remain intact. (This prevents accidental misreading of Data Stream Commands.) Attempting to read data as Data Stream Commands will have the same consequence.

The Input FIFO Channel addressing is controlled by the Input FIFO Read and Write Pointer SFRs. These SFRs are read only registers during normal operation. However, during Freeze Mode (See FIFO-External Host Interface Freeze Mode description), the internal CPU has write access to them. Any write to these registers in normal mode will have no effect. The Input Write Pointer SFR contains the address location to which data/commands are written from the Slave Bus Input/Slave Bus Command registers. The write pointer is automatically incremented after each write and is reset to zero if equal to the CBP, as the Input FIFO operates as a circular buffer.

If a write is performed on an empty FIFO, the first byte is also written into the FIFO IN or COMMAND IN SFR. If the Host continues writing while the Input FIFO is full, an external interrupt, if enabled, is sent to the host to signal the overrun condition. The writes are ignored by the FIFO control logic and the cycle is terminated. Similarly, an internal CPU read of an empty FIFO will cause an underrun error interrupt to be generated to the internal CPU and a value of "0FFH" will be read by the internal CPU.

The Read Pointer SFR holds the address of the next byte to be read from the Input FIFO. An Input FIFO read operation post-increments the Input Read Pointer SFR and loads a new data byte to the FIFO IN SFR or a Data Stream Command into the COMMAND IN SFR at the end of the read cycle.

A FIFO Request for Service (via DMA, Interrupt or a flag) is generated to the Host whenever more data can be written into the Input FIFO. For efficient utilization of the Host, a "threshold" value can be programmed into the Input FIFO Threshold SFR. The range of values of the Input FIFO Threshold SFR can be from 0 to (CBP-2). The Request for Service Interrupt is generated only after the Input FIFO has room to accommodate a threshold number of bytes or more. The threshold is equal to the total number of bytes in the Input FIFO minus the number of bytes programmed in the Input FIFO Threshold SFR. With this feature the Host is assured that it can write at least a threshold number of bytes to the Input FIFO channel without worrying about an overrun condition. Once the Request for Service is generated it remains active until the Input FIFO becomes full.

Output FIFO Channel

The Output FIFO Channel provides data transfer from the UPI-452 internal CPU to the external Host (Figure 6).

The registers associated with the Output Channel during normal operation are listed in Table 2*.

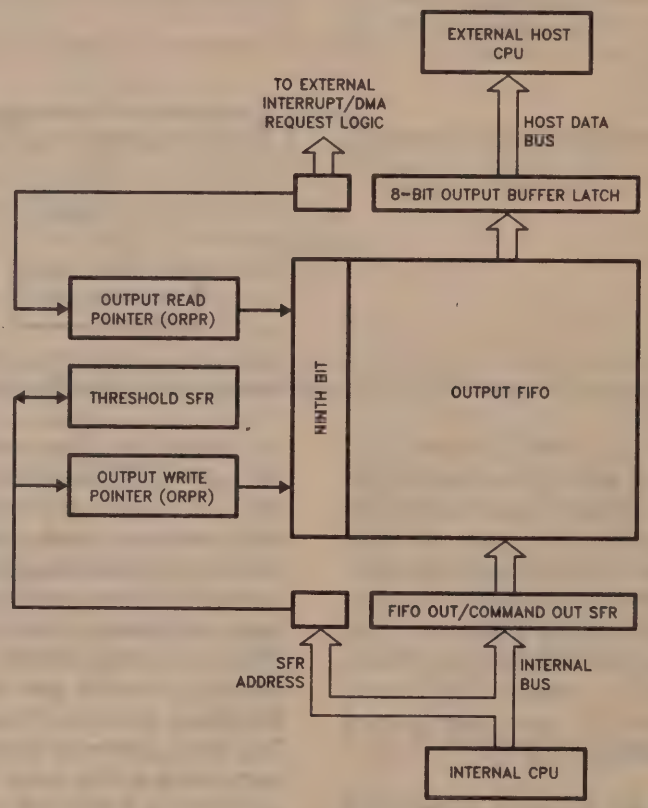


Figure 6. Output FIFO Channel Functional Block Diagram

Table 2. Output FIFO Channel Registers

	Register Name	Description
1)	Output Buffer Latch	Host CPU Read only
2)	FIFO OUT SFR	Internal CPU Read and Write
3)	COMMAND OUT SFR	Internal CPU Read and Write
4)	Output FIFO Read Pointer SFR	Internal CPU Read only
5)	Output FIFO Write Pointer SFR	Internal CPU Read only
6)	Output FIFO Threshold SFR	Internal CPU Read only

*See "FIFO-EXTERNAL HOST INTERFACE FREEZE MODE" section for Freeze Mode register characteristics description.

The UPI-452 internal CPU transfers data to the Output FIFO via the FIFO OUT SFR and commands via the COMMAND OUT SFR. If the byte is written to the COMMAND OUT SFR, the ninth bit is automatically set (=1) to indicate a Data Stream Command. If the byte is written to the FIFO OUT SFR the ninth bit is cleared (=0). Thus the FIFO OUT and COMMAND OUT SFRs are the same but the address determines whether the byte entered in the FIFO is a DSC or data byte.

The Output FIFO preloads a byte into the Output Buffer Latch. When the Host issues a RD/ signal, the data is immediately read from the Output Buffer Latch. The next data byte is then loaded into the Output Buffer Latch and an interrupt, if enabled, is generated if the byte is a DSC (ninth bit is set). The operation is carefully timed such that an interrupt can be generated in time for it to be recognized by the Host before its next read instruction. Internal CPU write and external Host read operations are interleaved at the FIFO so that they appear to be occurring concurrently.

The Output FIFO read and write pointer operation is the same as for the Input Channel. Writing to the FIFO OUT or COMMAND OUT SFRs will increment the Output Write Pointer SFR but reading from it will leave the write pointer unchanged. A rollover of the Output FIFO Write Pointer causes the pointer to be reset to the value in the Channel Boundary Pointer (CBP) SFR.

If the external host attempts to read a Data Stream Command as a data byte it will result in invalid data being read. The DSC is not lost because the invalid read does not increment the pointer. Similarly attempting to read a data byte as a Data Stream Command has the same result.

A Request for Service is generated to the external Host under the following two conditions:

1.) Whenever the internal CPU has written a threshold number of bytes or more into the Output FIFO (threshold = (OTHR) + 1). The threshold number should be chosen such that the bus latency time for the external Host does not result in a FIFO overrun error condition on the internal CPU side. The threshold limit should be large enough to make a bus request by the UPI-452 to the external host CPU worthwhile. Once a request for service is generated, the request remains active until the Output FIFO becomes empty. The range of values of the FIFO Output Threshold (OTHR) SFR is from 1 to the Output FIFO Size. The threshold number can be programmed via the OTHR SFR.

2.) The second type of Request for Service is called "Flush Mode" and occurs when the internal CPU writes a Data Stream Command into the Output FIFO. Its purpose is to ensure that a data block entered into the Output FIFO, which is less than the programmed threshold, will generate a Request for Service interrupt, if enabled, and be read, or "Flushed" from the Output FIFO, by the external host CPU regardless of the status of the OTHR SFR.

Immediate Commands

Immediate Commands provide direct communication between the external Host and UPI-452. Unlike Data Stream Commands which are entered into the FIFO, the Immediate Command is available to the receiving CPU directly, bypassing the FIFO. The Immediate Command can serve as a program vector pointing into a jump table in the recipients software. Immediate Command Interrupts are generated, if enabled, and a bit in the appropriate Status Register is set when an Immediate Command is input or output. A similar bit is provided to acknowledge when an Immediate Command has been read and whether the register is available to receive another command. The bits are reset when the Immediate Commands are read. Two Special Function Registers are dedicated to the Immediate Command interface. External addressing determines whether the Host is accessing the Input FIFO or the Immediate Command IN (IMIN) SFR. The internal CPU writes Immediate Commands to the Immediate Command OUT (IMOUT) SFR.

Both processors have the ability to enable or disable Immediate Command Interrupts. By disabling the interrupt, the recipient of the Immediate Command can poll the status SFR and read the Immediate Command at its convenience. Immediate Commands should only be written when the appropriate Immediate Command SFR is empty (as indicated in the appropriate status SFR:HCON/SCON). Similarly, the Immediate Command SFR should only be read when there is data in the Register.

The flowcharts in Figure 7a and 7b illustrate the proper handshake mechanisms between the external Host and internal CPU when handling Immediate Commands.

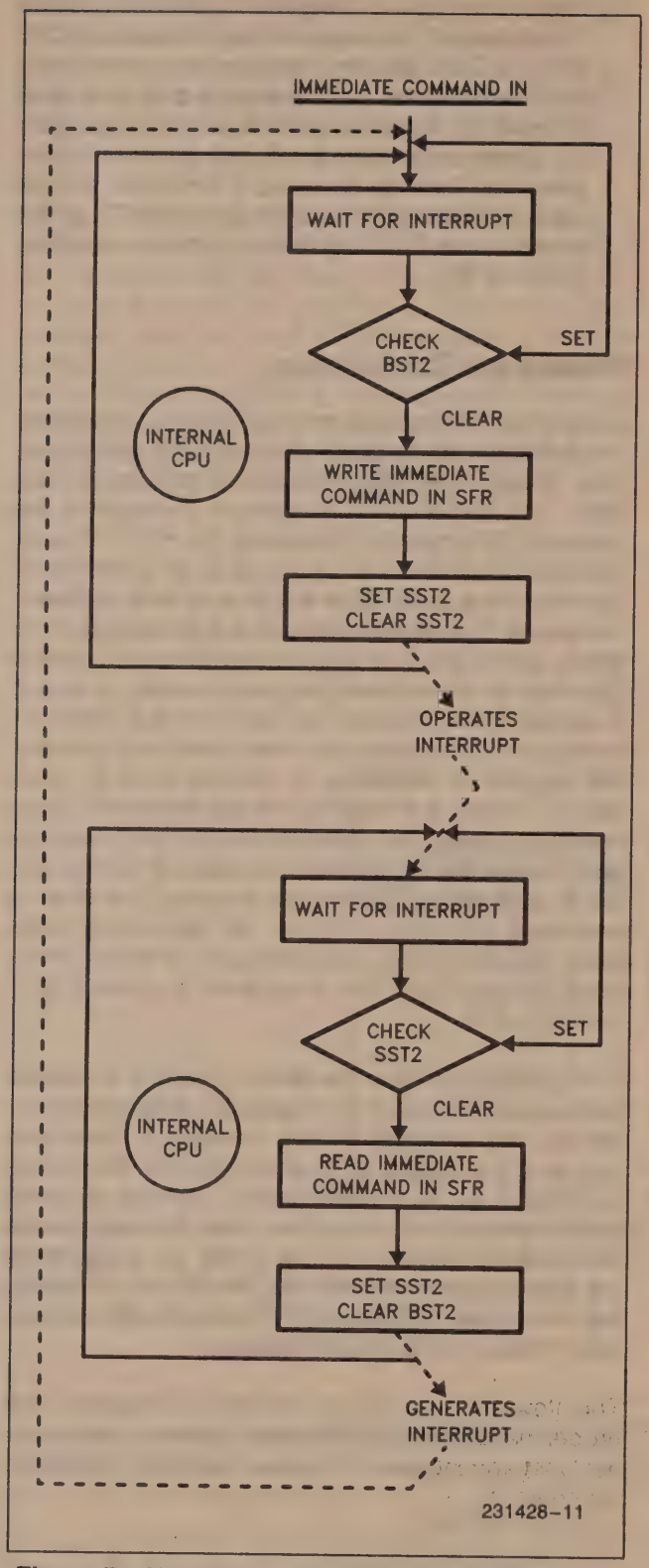


Figure 7a. Handshake Mechanisms for Handling Immediate Command IN Flowchart

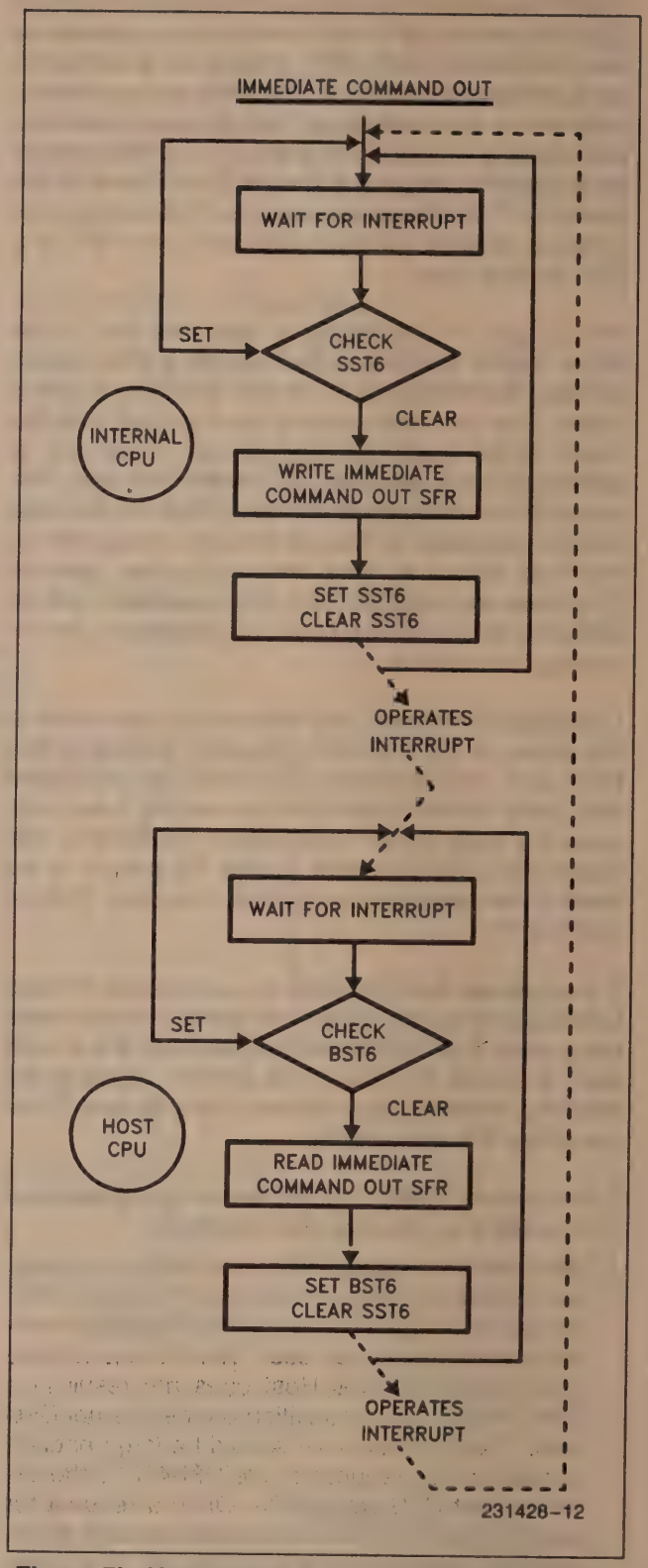


Figure 7b. Handshake Mechanisms for Handling Immediate Command OUT Flowchart

HOST & SLAVE INTERFACE SPECIAL FUNCTION REGISTERS

Slave Interface Special Function Registers

The Internal CPU interfaces with the FIFO slave module via the following registers:

1) Mode Special Function Register (MODE)
2) Slave Control Special Function Register (SLCON)
3) Slave Status Special Function Register (SSTAT)

Each register resides in the SFR Array and is accessible via all direct addressing modes except bit.

1) MODE Special Function Register (MODE)

The MODE SFR provides the primary control of the external host-FIFO interface. It is included in the SFR Array so that the internal CPU can configure the external host-FIFO interface should the user decide that the UPI-452 slave initialize itself independent of the external host CPU.

The MODE SFR can be directly modified by the internal CPU through direct address instructions. It can also be indirectly modified by the external host CPU by setting up a MODE SFR service routine in the UPI-452 program memory and having the host issue a Command, either Immediate or DSC, to vector to that routine.

Symbolic Address									Physical Address
MODE	—	MD6	MD5	MD4	—	—	—	—	0F9H
	(MSB)							(LSB)	
	Status On Reset:								
	1*	0	0	0	1*	1*	1*	1*	

MD7 (reserved)**

MD6 Request for Serivce to external CPU via;

1 = DMA (DRQIN/DRQOUT) request to external host when the Input or Output FIFO channel requests service

0 = Interrupt (INTRQIN/INTRQOUT or INTRQ) to external host when the Input or Output FIFO channel requests service or a DSC is encountered in the I/O Buffer Latch

MD5 Configure DRQIN/INTRQIN and DRQOUT/INTRQOUT to be either;

1 = Actively driven in both directions

0 = Open drain (tri-state)

MD4 Configure INTRQ to be either;

1 = Actively driven in both directions

0 = Open drain (tri-state)

MD3 (reserved) **

MD2 (reserved) **

MD1 (reserved) **

MD0 (reserved) **

2) Slave Control SFR (SLCON)

The Slave Control SFR is used to configure the FIFO-internal CPU interface. All interrupts are to the internal CPU.

Symbolic Address									Physical Address
SLCON	IFI	OFI	ICII	ICOI	FRZ	—	IFRS	OFRS	0E8H
	(MSB)							(LSB)	
	Status On Reset:								
	0	0	0	0	0	1*	0	0	

IFI Enable Input FIFO Interrupt (due to Underrun Error Condition, Data Stream Command or Request Service)

1 = Enable

0 = Disable

OFI Enable Output FIFO Interrupt (due to Overrun Error Condition or Requet Service)

1 = Enable

0 = Disable

Note: If the DMA is configured to serivce a FIFO demand, then the Request for Service Interrupt is not generated.

ICII Generate Interrupt when a command is written to the Immediate Command in Register

1 = Enable

0 = Disable

ICOI Generate Interrupt when Immediate Command Out Register is Available

1 = Enable

0 = Disable

FRZ Enable Freeze Mode

1 = Normal operation

0 = Freeze Mode

SC2 (reserved) **

IFRS Type of Input FIFO Channel Request for Service

1 = Request when Input FIFO not empty

0 = Request when Input FIFO full

OFRS Type of Output FIFO Channel Request for Service

1 = Request when Output FIFO not full

0 = Channel Request when Output FIFO empty

3) Slave Status SFR (SSTAT)

The bits in the Slave Status SFR reflect the status of the FIFO-internal CPU interface. It can be read during an internal interrupt service routine to determine the nature of the interrupt or read during a polling sequence to determine a course of action.

Symbolic Address									Physical Address
SSTAT	SST7	SST6	SST5	SST4	SST3	SST2	SST1	SST0	0E9H
	← Output FIFO Status →				← Input FIFO Status →				
	Status On Reset:								
	1	0	0	0	1	1	1	1	
	(MSB)							(LSB)	

SST7 Output FIFO Overrun Error Condition
- 1 = No Error
- 0 = Error (latched until Slave Status SFR is read)

SST6 Immediate Command Out Register Status
- 1 = Full (i.e. Host CPU has not read previous Immediate Command Out sent by internal CPU)
- 0 = Available

SST5 Freeze Mode Status
- 1 = Normal Operation
- 0 = Freeze Mode in Progress

SST4 Output FIFO Request for Service Flag
- 1 = Output FIFO does not request service
- 0 = Output FIFO requests service

SST3 Input FIFO Underrun Error Condition Flag
- 1 = No Underrun Error
- 0 = Underrun Error (latched until Slave Status SFR is read)

SST2 Immediate Command In SFR Status
- 1 = Empty
- 0 = Immediate Command received from host CPU

SST1 Data Stream Command/Data at Input FIFO Flag
- 1 = Data (not DSC)
- 0 = DSC (at COMMAND IN SFR)
- (Note: Only if SST0=0, if SST0=1 then undetermined)

SST0 Input FIFO Request For Service Flag
- 1 = Input FIFO Does Not Request Service
- 0 = Input FIFO Request for Service

NOTES:

*A '1' will be read from a SFR reserved location.
**'reserved'—these locations are reserved for future use by Intel Corporation.

EXTERNAL HOST INTERFACE SPECIAL FUNCTION REGISTERS

The external host CPU has direct access to the following SFRs:

1) Host Control Special Function Register
2) Host Status Special Function Register

It can also access other SFRs by commanding the internal CPU to change them accordingly via Data Stream Commands or Immediate Commands. The protocol for implementing this is entirely determined by the user.

1) Host Control SFR (HCON)

By writing to the Host Control SFR, the host can enable or disable FIFO interrupts and DMA requests and can reset the UPI-452.

Symbolic Address									Physical Address
HCON	HC7	HC6	HC5	HC4	HC3	—	HC1	—	0E7H
	(MSB)							(LSB)	
	Status On Reset:								
	0	0	0	0	0	1*	0	1*	

HC7 Enable Output FIFO Interrupt due to Underrun Error Condition, Data Stream Command or Service Request

1 = Enable

0 = Disable

HC6 Enable Input FIFO Interrupt due to Overrun Error Condition, or Service Request

1 = Enable

0 = Disable

HC5 Enable the generation of the Interrupt due to Immediate Command Out being present

1 = Enable

0 = Disable

HC4 Enable the Interrupt due to the Immediate Command in Register being Available for a new Immediate Command byte

1 = Enable

0 = Disable

HC3 Reset UPI-452

1 = Software RESET

0 = Normal Operation

HC2 (reserved) **

HC1 Select between INTRQ and INTRQIN/INTRQOUT as Request for Service interrupt signal when DMA is disabled

1 = INTRQ

0 = INTRQIN or INTRQOUT

HC0 (reserved) **

2) Host Status SFR (HSTAT)

The Host Status SFR provides information on the FIFO-Host Interface and can be used to determine the source of an external interrupt during polling. Like the Slave Status SFR, the Host Status SFR reflects the current status of the FIFO-external host interface.

Symbolic Address									Physical Address
HSTAT	HST7	HST6	HST5	HST4	HST3	HST2	HST1	HST0	0E6H
	← Output FIFO Status →				← Input FIFO Status →				
	Status On Reset:								
	1	1	1	1	1	0	0/1*	1	
	(MSB)							(LSB)	

HST7 Output FIFO Underrun Error Condition

1 = No Underrun Error

0 = Underrun Error (latched until Host Status Register is read)

HST6 Immediate Command Out SFR Status

1 = Empty

0 = Immediate Command Present

HST5 Data Stream Command/Data at Output FIFO Status

1 = Data (not DSC)

0 = DSC (present at Output FIFO COMMAND OUT SFR)

(Note: Only if HST4=0, if HST4=1 then undetermined)

HST4 Output FIFO Request for Service Statue

1 = No Request for Service

0 = Output FIFO Request for Service due to;

a. Output FIFO containing the threshold number of bytes or more

b. Internal CPU sending a block of data terminated by a DSC (DSC alone clears upon being read)

HST3 InPut FIFO Overrun Error Condition

1 = No Overrun Error

0 = Overrun Error (latched until Host Status Register is read)

HST2 Immediate Command In SFR Status

1 = Full (i.e. Internal CPU has not read previous Immediate Command sent by Host)

0 = Empty

Reset value;

'0' - if read by the external Host

'1' - if read by internal CPU (reads shadow latch - see Freeze Mode description)

HST1 Freeze Mode Status

1 = Freeze mode in progress.

(In freeze mode, the bits of the Host Status SFR are forced to a '1' initially to prevent the external Host from attempting to access the FIFO. The definition of the Host Status SFR bits during freeze mode can be found in Freeze Mode description)

0 = Normal Operation

HST0 Input FIFO Request Service Status

1 = Input FIFO does not request service

0 = Input FIFO request service due to the Input FIFO containing enough space for the host to write the threshold number of bytes or more

Note: * A '1' will be read from a SFR reserved location

**'reserved' - these locations are reserved for future use by Intel Corportion

FIFO MODULE - EXTERNAL HOST INTERFACE

Overview

The FIFO-external host interface supports asynchronous bi-directional 8-bit data transfers for a Host operating up to 10 MHz. The host interface is fully compatible with Intel microprocessor local busses and with MULTIBUS. The FIFO has two specialized DMA request pins for Input and Output FIFO channel DMA requests. These are multiplexed to provide a dedicated Request for Service interrupt (DRQIN/INTRQIN, DRQOUT/INTRQOUT).

The external Host can program, under user defined protocol, thresholds into the FIFO Input and Output Threshold SFRs which determine when the FIFO Request for Service interrupt is generated. The FIFO module external Host interface is configured by the internal CPU via the MODE SFR. The external Host can enable and disable the interfacing pins (INTRQ, DRQIN/INTRQIN and DRQOUT/INTRQOUT) via the Host Control SFR. Data Stream Commands in the Input FIFO channel allow the Host to influence the processing of data blocks and are sent with the data flow to maintain synchronization. Data Stream Commands in the Output FIFO Channel allow the internal CPU to perform the same function, and also to set the Output FIFO Request Service status logic to the host CPU regardless of the programmed value in the Threshold SFR.

Slave Interface Address Decoding

The UPI-452 determines the desired Host function through address decoding. The lower three bits of the address as well as the Read, Write, Chip Select and DMA Acknowledge are used for decoding. Table 3 shows the pin states and the Read or Write operations associated with each configuration.

Table 3. UPI-452 Address Decoding

$\overline{DACK}$	$\overline{CS}$	A2	A1	A0	Read	Write
1 1	1 0	X 0	X 0	X 0	No Operation Data or DMA from Output FIFO Channel	No Operation Data or DMA to Input FIFO Channel
1	0	0	0	1	Data Stream Command from Output FIFO Channel	Data Stream Command to Input FIFO Channel
1	0	0	1	0	Host Status SFR Read	Reserved
1	0	0	1	1	Host Control SFR Read	Host Control SFR Write
1	0	1	0	0	Immediate Command SFR Read	Immediate Command to SFR Write
1 0	0 X	1 X	1 X	X X	Reserved DMA Data from Output FIFO Channel	Reserved DMA Data to Input FIFO Channel

NOTES:
1. Attempting to read a DSC as a data byte will result in invalid data being read. The read pointers are not incremented so that the DSC is not lost. Attempting to read a data byte as a DSC has the same result.
2. If DACK/ is active the UPI-452 will attempt a DMA operation when RD/ or WR/ becomes active regardless of the DMA enable bit (MD6) in the MODE SFR. Care should be taken when using DACK/. For proper operation, DACK/ must be driven high (+5V) when not using DMA.

Interrupts to the Host

The UPI-452 interrupts the external Host via the INTRQ pin. In addition, the DRQIN and DRQOUT pins can be multiplexed as interrupt request lines, INTRQIN and INTRQOUT respectively, when DMA is disabled. This provides two special FIFO "Request for Service" interrupts.

There are six FIFO-related interrupt sources; two From The Input FIFO; three From The Output FIFO; and one from the Immediate Command Out SFR.

INPUT FIFO: The Input FIFO interrupt is generated whenever:

a. The Input FIFO contains space for a threshold number of bytes.

b. When an Input FIFO overrun error condition exists. The appropriate bits in the Host Status SFR are set and the interrupt is generated only if enabled.

OUTPUT FIFO: The Output FIFO Request for Service Interrupt operates in the same manner as the Input FIFO interrupt:

a. When the FIFO contains the threshold number of bytes or more.

b. Output FIFO error condition interrupts are generated when the Output FIFO is underrun.

c. There are also interrupts due to the presence of a Data Stream Command in the output FIFO.

A Data Stream Command interrupt is used to halt normal processing, using the command as a vector to a service routine. When DMA is disabled, the user may program (through HC1) INTRQ to include FIFO Request for Service Interrupts or use INTRQIN and INTRQOUT as Request for Service Interrupts.

IMMEDIATE COMMAND OUT SFR:

a. An Immediate Command Out Interrupt is generated when the internal CPU writes to the Immediate Command Out SFR. It allows the internal CPU to bypass the FIFO when communicating with the external Host.

b. An Immediate Command Interrupt is generated when the Immediate Command SFR is empty.

FREEZE MODE: When the internal CPU invokes FIFO Freeze Mode, for example at reset or to reconfigure the FIFO interface, INTRQ is activated. The INTRQ can only be deactivated by the external Host reading the Host Status SFR (HST1 remains active until Freeze Mode is disabled by the internal CPU).

Once an interrupt is generated, INTRQ will remain high until no interrupt generating condition exists. For a FIFO underrun/overrun error interrupt, the interrupt condition is deactivated by the external Host reading the Host Status SFR. An interrupt is serviced by reading the Host Status SFR to determine the source of the interrupt and vectoring the appropriate service routine.

DMA Requests to the Host

The UPI-452 generates two DMA requests, DRQIN and DRQOUT, to facilitate data transfer between the Host and the Input and Output FIFO channels. A DMA acknowledge, $\overline{DACK}$, is used as a chip select and initiates a data transfer. The external $\overline{READ}$ and $\overline{WRITE}$ signals select the Input and Output FIFO respectively. The $\overline{CS}$ and address lines can also be used as a DMA acknowledge for processors with onboard DMA controllers which do not generate a $\overline{DACK}$ signal.

The internal CPU can configure the UPI-452 to request service from the external host via DMA or interrupts by programming Mode SFR MD6 bit. In addition the external Host enabled DMA requests through bits 6 and 7 of the Host Control SFR. When a DMA request is invoked the number of bytes transferred to the Input FIFO is the total number of bytes in the Input FIFO (as determined by the CBP SFR) minus the value programmed in the Input FIFO Threshold SFR. The DMA request line is activated only when the Input FIFO has a threshold number of bytes that can be transferred.

The Output FIFO DMA request is activated when a DSC is written by the internal CPU at the end of a block of data (Flush Mode) or when the Output FIFO threshold is reached. The request remains active until the Input FIFO becomes full or the Output FIFO becomes empty. If a DSC is encountered the DMA request is dropped until the DSC is read. The DMA request will be reactivated after the DSC is read and remains active until the Output FIFO becomes empty or another DSC is encountered. When a block of data is being transferred via DMA and if a DSC is encountered, the Output FIFO DMA request will be automatically deactivated prior to a DSC being read out of the FIFO.

FIFO MODULE - INTERNAL CPU INTERFACE

Overview

The Input and Output FIFOs are accessed by the internal CPU through direct addressing of the FIFO IN/COMMAND IN and FIFO OUT/COMMAND OUT Special Function Registers. All of the 80C51 instructions involving direct addressing may be used to access the FIFO's SFRs. The FIFO IN, COMMAND IN and Immediate Command In SFRs are actually read only registers, and their Output counterparts are write only. Internal DMA transfers data between Internal memory, External Memory and the Special Function Registers. The Special Function Registers appear as another group of dedicated memory addresses and are programmed as the source or destination via the DMA0/DMA1 Source Address or Destination Address Special Function Registers. The FIFO module manages the transfer of data between the external host and FIFO SFRs.

Internal CPU Access to FIFO Via Software Instructions

The internal CPU has access to the Input and Output FIFOs via the FIFO IN/COMMAND IN and FIFO OUT/COMMAND OUT SFRs which reside in the Special Function Register Array. At the end of every instruction that involves a read of the FIFO IN/COMMAND IN SFR, the SFR is written over by a new byte from the Input FIFO channel when available. At the end of every instruction that involves a write to the FIFO OUT/COMMAND OUT SFR, the new byte is written into the Output FIFO channel and the write pointer is incremented after the write operation (post incremented).

The internal CPU reads the Input FIFO by using the FIFO IN/COMMAND IN SFR as the source register in an instruction. Those instructions which read the Input FIFO are listed below:

```
ADD  A,FIFO IN/COMMAND IN
ADDC A,FIFO IN/COMMAND IN
PUSH FIFO IN/COMMAND IN
ANL  A,FIFO IN/COMMAND IN
ORL  A,FIFO IN/COMMAND IN
XRL  A,FIFO IN/COMMAND IN
CJNE A,FIFO IN/COMMAND IN, rel
SUBB A,FIFO IN/COMMAND IN
MOV  direct,FIFO IN/COMMAND IN
MOV  @Ri,FIFO IN/COMMAND IN
MOV  Rn,FIFO IN/COMMAND IN
MOV  A,FIFO IN/COMMAND IN
```

After each access to these registers, they are overwritten by a new byte from the FIFO.

NOTE:
Instructions which use the FIFO IN or COMMAND IN SFR as both a source and destination register will have the data destroyed as the next data byte is rewritten into the FIFO IN register at the end of the instruction. These instructions are not supported by the UPI-452 FIFO. Data can only be read through the FIFO IN SFR and DSCs through the COMMAND IN SFR. Data read through the COMMAND IN SFR will be read as OFFH, and DSDs read through the FIFO IN SFR will be read as OFFH. The Immediate Command in SFR is read with the same instructions as the FIFO IN and COMMAND IN SFRs.

The FIFO IN, COMMAND IN and Immediate Command In SFRs are read only registers. Any write operation performed on these registers will be ignored and the FIFO pointers will remain intact.

The internal CPU uses the FIFO OUT SFR to write to the Output FIFO and any instruction which uses the FIFO OUT or COMMAND OUT SFR as a destination will invoke a FIFO write. DSCs are differentiated from data by writing to the COMMAND OUT SFR. In the FIFO, Data Stream Commands have the ninth bit associated with the command byte set to "1". The instructions used to write to the Output FIFO are listed below:

```
MOV FIFO OUT/COMMOUT, A
MOV FIFO OUT/COMMOUT, direct
MOV FIFO OUT/COMMOUT, Rn
POP FIFO OUT/COMMOUT
MOV FIFO OUT/COMMOUT, #data
MOV FIFO OUT/COMMOUNT, @Ri
```

NOTE:
Instructions which use the FIFO OUT/COMMAND OUT SFRs as both a source and destination register cause invalid data to be written into the Output FIFO. These instructions are not supported by the UPI-452 FIFO.

GENERAL PURPOSE DMA CHANNELS

Overview

There are two identical General Purpose DMA Channels on the UPI-452 which allow high speed data transfer from one writeable memory space to another. As many as 64K bytes can be transferred in a single DMA operation. The following memory spaces can be used with DMA channels:

- Internal Data Memory
- External Data Memory
- Special Function Registers

The Special Function Register array appears as a limited group of dedicated memory addresses. The Special Function Registers may be used in DMA transfer operations by specifying the SFR as the source of destination address. The Special Function Registers which may be used in DMA transfers are listed in Table 4. Table 4 also shows whether the SFR may be used as Source or Destination only, or both.

Table 4. DMA Accessible Special Function Registers

SFR	Symbol	Address	Source Only	Destination Only	Either
Accumulator	A/ACC	0E0H			Y
B Register	B	0F0H			Y
FIFO IN	FIN	0EEH	Y		
COMMAND IN	CIN	0EFH	Y		
FIFO OUT	FOUT	0FEH		Y	
COMMAND OUT	COUT	0FFH		Y	
Serial Data Buffer	SBUF	099H			Y
Port 0	P0	080H			Y
Port 1	P1	090H			Y
Port 2	P2	0A0H			Y
Port 3	P3	0B0H			Y
Port 4	P4	0C0H			Y

The FIFO can be accessed during DMA by using the FIFO IN SFR as the DMA Source Address Register (SAR) or the FIFO OUT SFR as the Destination Address Register (DAR). (Note: Since the FIFO IN SFR is a read only register, the DMA transfer will be ignored if it is used as a DMA DAR. This is also true if the FIFO OUT SFR is used as a DMA SAR.)

Each DMA channel is software programmable to operate in either Block Mode or Demand Mode. In the Block Mode, DMA transfers can be further programmed to take place in Burst Mode or Alternate Cycle mode. In Burst Mode, the processor halts its execution and dedicates its resources to the DMA transfer. In Alternate Cycle Mode, DMA cycles and instruction cycles occur alternately.

In Demand Mode, a DMA transfer occurs only when it is demanded. Demands can be accepted from an external device (through External Interrupt pins, EXT0/EXT1) or from either the Serial Channel or FIFO flags. In this way, a DMA transfer can be synchronized to an external device, the FIFO or the Serial Port. If the External Interrupt is configured in Edge Mode, a single byte transfer occurs per transition. The external interrupt itself will occur if enabled. If the External Interrupt is configured in Level Mode, DMA transfers continue until the External Interrupt request goes inactive or the byte count becomes zero. The following flags activate Demand Mode transfers of one byte to/from the FIFO or Serial Channel:

RI - Serial Channel Receiver Buffer Full

TI - Serial Channel Transmitter Buffer Empty

DIFRS - Input FIFO Request Service

DOFRS - Output FIFO Request Service

(DIFRS differs from bit 0 of the Slave Status SFR (SST0 - Input FIFO Request Service flag) in that it is deactivated when a DSC is to be read from the Input FIFO.)

Architecture

There are three 16 bit and one 8 bit Special Function Registers associated with each DMA channel.

- The 16 bit Source Address SFR (SAR) points to the source byte.
- The 16 bit Destination Address SFR (DAR) points to the destination.
- The 16 bit Byte Count SFR (BCR) contains the number of bytes to be transferred and is decremented when a byte transfer is accomplished.
- The DMA Control SFR (DCON) is eight bits wide and specifies the source memory space, destination memory space and the mode of operation.

In Auto Increment mode, the Source Address and/or Destination Address is incremented when a byte is transferred. When a DMA transfer is complete (BCR = 0), the DONE bit is set and a maskable interrupt is generated. The GO bit must be set to start any DMA transfer (also, the Slave Control SFR FRZ bit must be set to disable Freeze Mode). The two DMA channels are designated as DMA0 and DMA1, and their corresponding registers are suffixed by 0 or 1; e.g. SAR0, DAR1, etc. To transfer 64K bytes of data the BCR should be programmed to zero.

DMA Special Function Registers

DMA Control SFR: DCON0, DCON1

Symbolic Address									Physical Address
DCON0	DAS	IDA	SAS	ISA	DM	TM	DONE	GO	092H
DCON1	DAS	IDA	SAS	ISA	DM	TM	DONE	GO	093H
	(MSB)							(LSB)	

Reset Status: DCON0 and DCON1 = 00H

Bit Definition:

DAS	IDA	Destination Address Space
0	0	External Data Memory without Auto-Increment
0	1	External Data Memory with Auto-Increment
1	0	Special Function Register
1	1	Internal Data Memory

SAS	ISA	Source Address Space
0	0	External Data Memory without Auto-Increment
0	1	External Data Memory with Auto-Increment
1	0	Special Function Register
1	1	Internal Data Memory

DM	TM	DMA Transfer Mode
0	0	Alternate-Cycle Transfer Mode
0	1	Burst Transfer Mode
1	0	FIFO or Serial Channel Demand Mode
1	1	External Demand Mode

DONE DMA transfer Flag:

0 DMA transfer is not completed.

1 DMA transfer is complete.

NOTE:
This flag is set when contents of the Byte Count SFR decrements to zero. It is reset automatically when the DMA vectors to its interrupt routine.

GO Enable DMA Transfer:

0 Disable DMA transfer (in all modes).

1 Enable DMA transfer. If the DMA is in the Block mode, start DMA transfer if possible. If it is in the Demand mode, enable the channel and wait for a demand.

NOTE:
The GO bit is reset when the BCR decrements to zero.

DMA Transfer Modes

The following five modes of DMA operation are possible in the UPI-452.

BURST MODE

In BURST mode the DMA is initiated by setting the GO bit in the DCON SFR. The DMA operation continues until BCR decrements to zero (zero byte count), then an interrupt is generated (if enabled). No interrupts are recognized during a DMA operation once started.

INPUT CHANNEL:The FIFO Input Channel can be used in burst mode by specifying the FIFO IN SFR as the DMA Source Address. DMA transfers begin when the GO Bit in the DMA Control SFR is set. The number of bytes to be transferred must be specified in the Byte Count SFR (BCR) and auto-incrementing of the SAR must be disabled. Once the GO bit is set nothing can interrupt the transfer of data until the BCR is zero. In this mode, a Data Stream Command encountered in the FIFO will be held in the COMMAND IN SFR with the pointers frozen, and invalid data (FFH) will be read through the FIFO IN SFR. If the Input FIFO becomes empty during the block transfer, an OFFH will be read until BCR decrements to zero.

OUTPUT CHANNEL:The Output FIFO Channel can be used in burst mode by specifying the FIFO OUT or COMMAND OUT SFR as the DMA Destination Address. DMA transfers begin when the GO bit is set. This mode can be used to send a block of data or a block of Data Stream Commands. If the FIFO becomes full during the block transfer, the remaining data will be lost.

(Note: All interrupts including FIFO interrupts are not recognized in Burst Mode. Burst Mode transfers should be used to service the FIFO only when the user is certain that no Data Stream Commands are in the block to be transferred (Input FIFO) and that

the FIFO contains enough space to store the block to be transferred. In all other cases Alternate Cycle or Demand Mode should be used.)

2. ALTERNATE CYCLE MODE

Alternate cycle mode is useful when CPU processing must occur during the DMA transfers. In this mode, a DMA cycle and an instruction cycle occur alternately. The interrupt request is generated (if enabled) at the end of the process, i.e. when BCR decrements to zero. The transfer is initiated by setting the GO bit in the DCON SFR.

3. EXTERNAL DEMAND MODE

The DMA can be initiated by an external device via External Interrupt 0 and 1 (INT0/EINT1) pins. The INTO pin demands DMA0 (Channel 0) and INT1 demands DMA1 (Channel 1). If the interrupts are configured in edge mode, a single byte transfer is accomplished for every request. Interrupts also result (INT0 or INT1) after every byte transfer (if enabled). If the interrupts are configured in level mode, the DMA transfer continues until the request goes inactive or BCR=0. In either case, a DMA interrupt is generated (if enabled) when BCR=0. The GO bit must be set for the transfer to begin.

4. FIFO DEMAND AND ALTERNATE CYCLE DEMAND MODES

Although any DMA mode is possible using the FIFO buffer, only Demand and Alternate Cycle Demand Modes make sense. Demand Mode DMA transfers using the Input FIFO Channel are set-up by setting the GO Bit and specifying the FIFO IN register as the DMA Source Address Register. The BCR should be set to the maximum number of expected transfers. The user must also program bit 1 of the Slave Control Register (SC1) to determine whether the FIFO Request For Service Flag will be set when the FIFO becomes not empty or full. Once the Request For Service Flag is set by the FIFO, the DMA transfer begins, and continues until the request flag is deactivated. While the request is active, nothing can interrupt the DMA (i.e. it behaves like burst mode). The DMA Request is held active until one of the following occurs:

1) The FIFO becomes empty
2) A Data Stream Command is encountered (this generates a FIFO interrupt and DMA operation resumes after the Data Stream command is read.)
3) BCR = 0 (this generates a DMA interrupt and sets the DONE Bit)

DMA transfers to the Output FIFO Channel are similar. The FIFO OUT or COMMAND OUT SFR is the DMA Destination Address SFR and a transfer is started by setting the GO bit. The user programs bit 0 of the Slave Control SFR (SC0) to determine whether a demand occurs when the Output FIFO is not full or empty. DMA transfers begin when the Request For Service Flag is set by the FIFO logic and continue as long as the flag is set. The Flag remains set until one of the following occurs:

1) The FIFO becomes full
2) BCR = 0 (this generates a DMA interrupt and sets the DONE bit)

Alternate cycle demand mode is also useful for FIFO transfers of a less urgent nature. As mentioned before, CPU instruction cycles are interleaved with DMA transfer cycles, allowing true parallel processing.

This mode differs from FIFO Demand Mode in that CPU instruction cycles must be interleaved with DMA transfers, even if the FIFO is demanding DMA. In FIFO Demand Mode, CPU cycles would never occur if the FIFO demand was present.

In either mode, the FIFO logic resets the interrupt flag after transferring the byte, so the interrupt is never generated.

5. SERIAL PORT DEMAND MODE

Demand mode is the logical choice when using the Serial Port. The DMA's can be activated by one of the Serial Channel Flags, Receiver Interrupt (RI) or Transmitter Interrupt (TI).

After the GO bit is set, the DMA is activated if one of the following conditions takes place;

SARO = SBUF and RI flag is set
DARO = SBUF and TI flag is set
SARO = FIFO In and IFRS flag is set
DARO = FIFO OUT and OFRS flag is set

NOTE:
TI flag must be set by software to initiate the first transfer.

When the DMA transfer begins, only one byte is transferred at a time. The serial port hardware automatically resets the flag after completion of the transfer, so an interrupt will not be generated. The DMA interrupt (if enabled) is not generated until BCR=0.

EXTERNAL MEMORY DMA:

When transferring data to or from external memory via DMA, the HOLD (HLD) and HOLD-ACKNOWLEDGE (HLDA) signals are used for handshaking. The HOLD and HOLD-ACKNOWLEDGE are active low signals which arbitrate control of the local bus. The UPI-452 can be used in a system where multi-masters are connected to a single parallel Address/Data bus. The HLD/HLDA signals are used to share resources (memory, peripherals, etc.) among all the processors on the local bus. The UPI-452 can be configured in any of three different External Memory Modes controlled by bits 5 and 6 (REQ & ARB) in the PCON SFR (Table 5). Each mode is described below:

REQUESTER MODE: In this mode, the UPI-452 is not the bus master, but must request the bus from another device. The UPI-452 configures port pin PI.6 as a HLD output and pin PI.7 as a HLDA input. The UPI-452 issues a HLD signal when it needs external access for a DMA channel. It uses the local bus after receiving the HLDA signal from the bus master, and will not release the bus until its DMA operation is complete.

ARBITER MODE: In this mode, the UPI-452 is the bus master. It configures port pin PI.6 as HLD input and pin PI.7 as HLDA output. When a device asserts the HLD signal to use the local bus, the UPI-452 asserts the HLDA signal after current instruction execution is complete. If the UPI-452 needs an external access via a DMA channel, it waits until the requester releases the bus, HLD goes inactive.

DISABLE (NON-DMA) MODE: When external program memory is accessed by an instruction or by program counter overflow beyond the internal ROM address, or when external data memory is assessed by MOVX instructions, the HLD/HLDA sequence is not initiated, since this is not a DMA memory access.

The balance of the PCON SFR bits are described in the "80C51 Register Description:Power Control SFR" section below.

Latency

When the GO bit is set, the UPI-452 finishes the current instruction before starting the DMA operation. Thus the maximum latency is 3.0 microseconds (at 16 MHz).

DMA Interrupt Vectors

Each DMA channel has a unique vectored interrupt associated with it. There are two vectored interrupts associated with the two DMA channels. The DMA interrupts are enabled and priorities set via the Interrupt Enable and Priority SFR (see "Interrupts" section). The interrupt priority scheme is similar to the scheme in 80C51.

When a DMA operation is complete (BCR decrements to zero), the DONE flag in the respective DCON (DCON0 or DCON1) SFR is set. If the DMA interrupt is enabled, the DONE flag is reset automatically upon vectoring to the interrupt routine.

Interrupts When DMA is Active

If a Burst Mode DMA transfer is in progress, the interrupts are not serviced until the DMA transfer is complete. This is also true for level activated External Demand DMA transfers. During Alternate Cycle DMA transfers, however, the interrupts are serviced at the end of the DMA cycle. After that, DMA cycles and instruction execution cycles occur alternately. In the case of edge activated External Demand Mode DMA transfers, the interrupt is serviced at the end of DMA transfer of that single byte.

Table 5. DMA MODE CONTROL - PCON SFR

Symbolic Address									Physical Address
PCON	—*	ARB	REQ	—*	—*	—*	—*	—*	87H
	(MSB)							(LSB)	

*Defined as per MLS-51 Data Sheet
Reset Status: 00H

Definition:

ARB	REQ	
0	0	HLD/HLDA logic is disabled.
0	1	The UPI-452 is in the Requester Mode.
1	0	The UPI-452 is in the Arbiter Mode.
1	1	Invalid

DMA Arbitration

Only one of the two DMA channels is active at a time, except when both are configured in the Alternate Cycle mode. In this case, the DMA cycles and Instruction Execution cycles occur in the following order:

1. DMA Cycle 0.
2. Instruction execution.
3. DMA Cycle 1.
4. Instruction execution.

DMA0 has priority over DMA1 during simultaneous activation of the two DMA channels. If one DMA channel is active, the other DMA channel, if activated, waits until the first one is complete.

If DMA0 is already in the Alternate Cycle mode and DMA1 is activated in Alternate Cycle Mode, it will take two instruction cycles before DMA1 is activated (due to the priority of DMA0). Once DMA1 becomes active, the execution will follow the normal sequence.

If DMA0 is already in the Alternate Cycle mode and DMA1 is activated in Burst Mode, the DMA1 Burst transfer will follow the DMA0 Alternate Cycle transfer (after the completion of the next instruction).

If the UPI-452 (as a Requester) asserts a HLD signal to request a DMA transfer (see "External Memory DMA")and its other DMA Channel requests a transfer before the HLDA signal is received, the channel having higher priority is activated first.

If, while executing a DMA transfer, the Arbiter receives a HLD signal, and then before it can acknowledge, its other DMA Channel requests a transfer, it then completes the second DMA transfer before sending the HLDA signal to release the bus to the HLD request.

The DMA Transfer waveforms are in Figures 8-11.

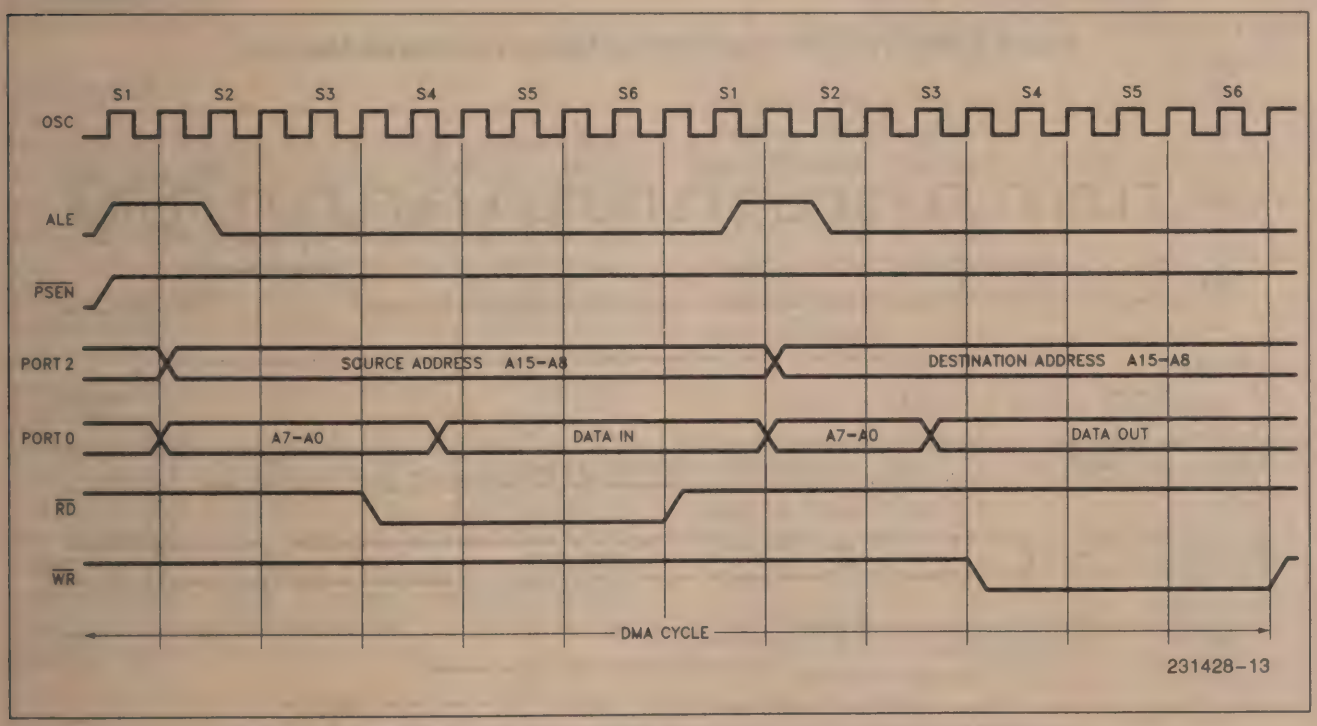


Figure 8. DMA Transfer from External Memory to External Memory

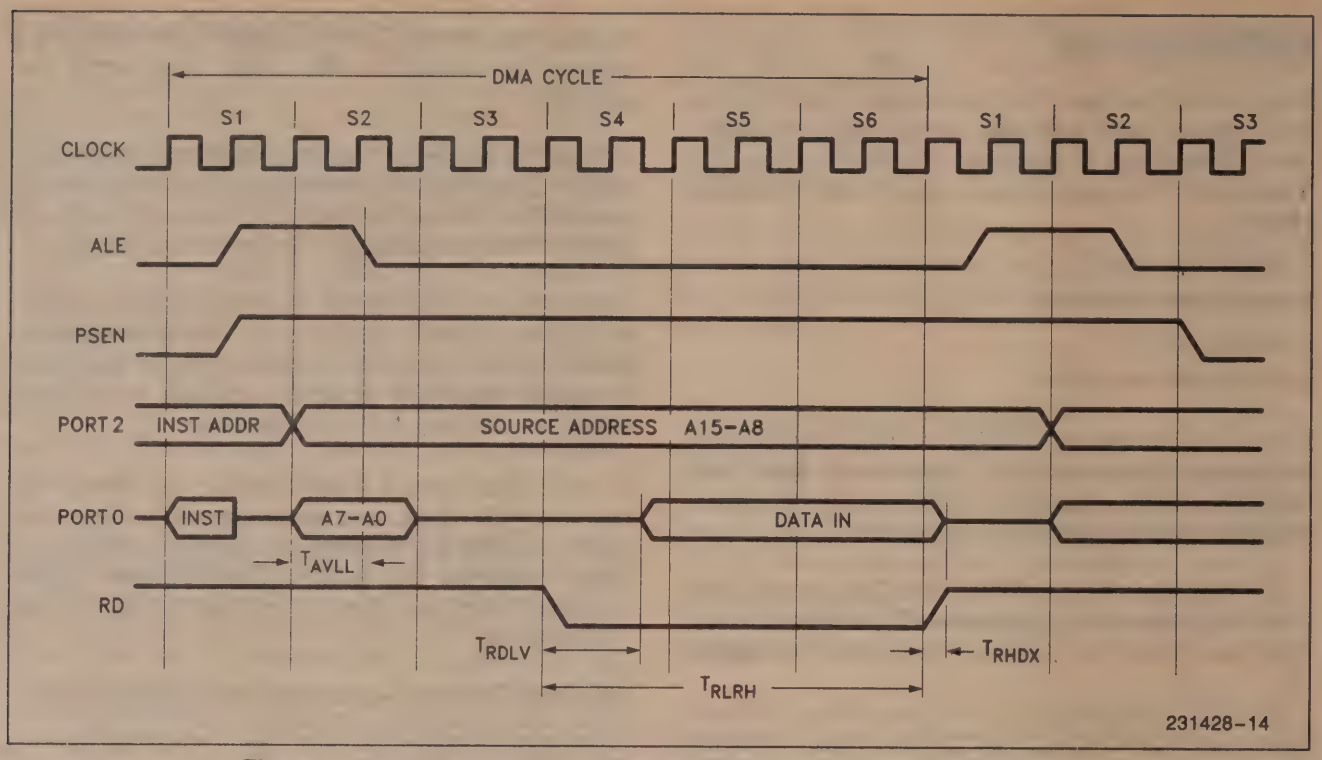


Figure 9. DMA Transfer from External Memory to Internal Memory

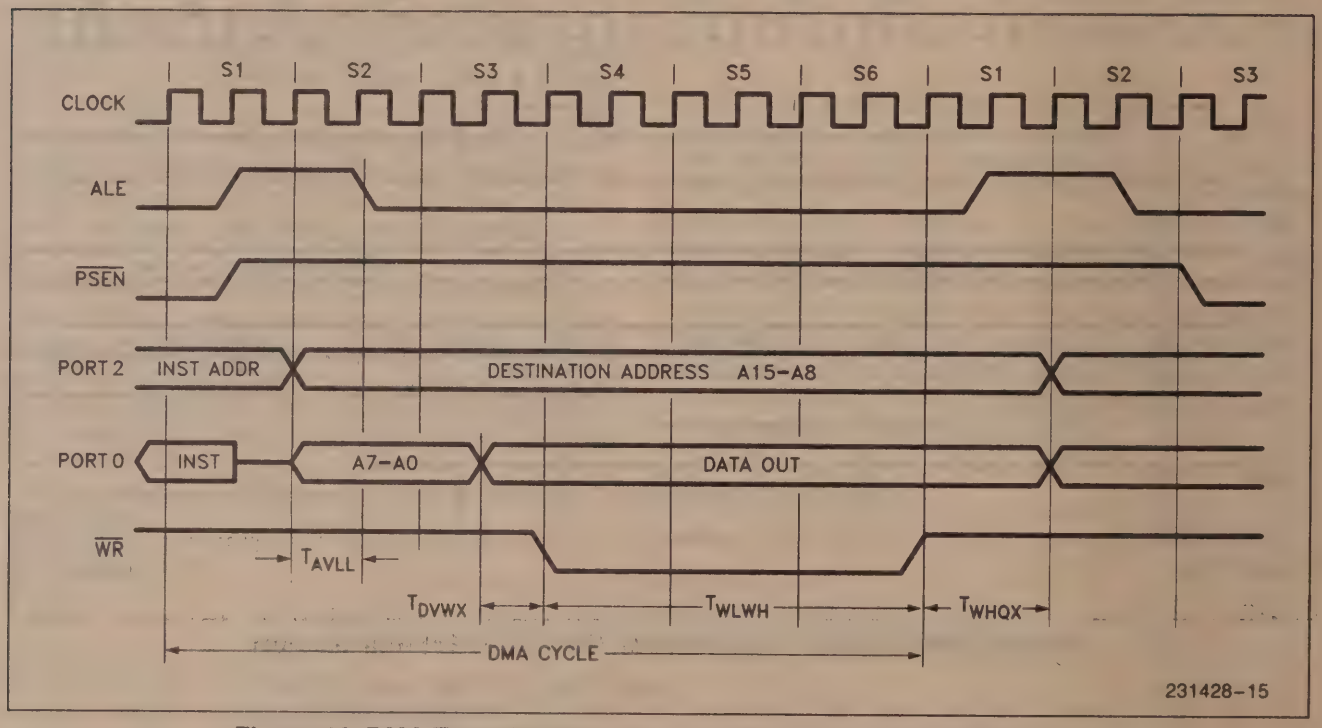


Figure 10. DMA Transfer from Internal Memory to External Memory

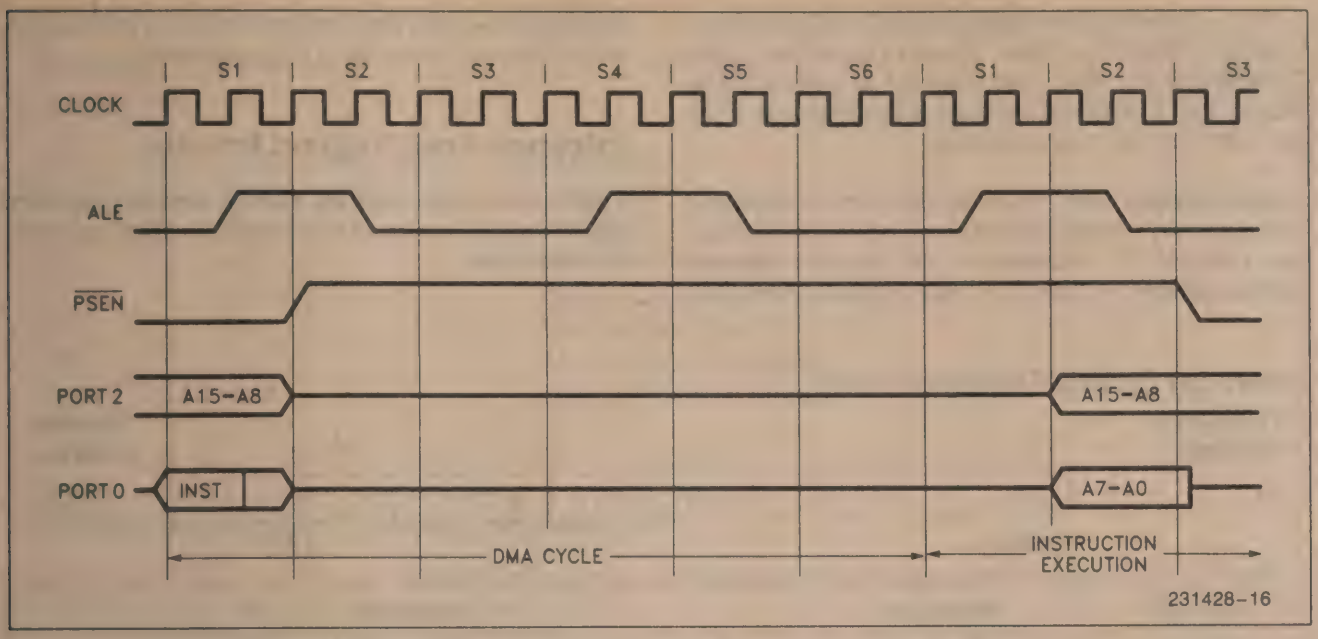


Figure 11. DMA Transfer from Internal Memory to Internal Memory

INTERRUPTS

Overview

The UPI-452 provides eight interrupt sources (Table 6). Their operation is the same as in the 80C51, with the addition of new interrupt sources for the UPI-452 FIFO and DMA features. These added interrupts have their enable and priority bits in the Interrupt Enable and Priority (IEP) SFR. The IEP SFR is in addition to the 80C51 Interrupt Enable (IE) and Interrupt Priority (IP) SFRs. The added interrupt sources are also globally enabled or disabled by the EA bit in the Interrupt Enable SFR. Table 6 lists the eight interrupt sources in order of priority. Table 7 lists the eight interrupt sources and their respective address vector location in program memory. (DMA interrupts are discussed in the "General Purpose DMA Channels" section. Additional interrupt information for Timer/Counter, Serial Channel, External Interrupt may be found in the Microcontroller Handbook for the 80C51.)

Table 6. Interrupt Priority

Interrupt Source	Priority Level
	(highest)
External Interrupt 0	0
Internal Timer/Counter 0	1
DMA Channel 0 Request	2
External Interrupt 1	3
DMA Channel 1 Request	4
Internal Timer/Counter 1	5
FIFO - Slave Bus Interface Buffer	6
Serial Channel	7
	(lowest)

Table 7. Interrupt Vector Addresses

Interrupt Source	Starting Address
External Interrupt 0	3 (003H)
Internal Timer/Counter 0	11 (00BH)
External Interrupt 1	19 (013H)
Internal Timer/Counter 1	27 (01BH)
Serial Channel	35 (023H)
FIFO - Slave Bus Interface Buffer	43 (02BH)
DMA Channel 0 Request	51 (033H)
DMA Channel 1 Request	59 (03BH)

FIFO Module Interrupts to Internal CPU

The FIFO module generates interrupts to the internal CPU whenever the FIFO requests service or when a Data Stream Command is in the COMMAND IN SFR. The Input FIFO will request service whenever it becomes full or not empty depending on bit 1 of the Slave Control SFR (IFRS). Similarly, the Output FIFO requests service when it becomes empty or not full as determined by bit 0 of the Slave Control SFR (OFRS). Request for Service interrupts are generated only if enabled by the internal CPU and if DMA requests are disabled via the MODE SFR and the Interrupt Enable SFR.

A Data Stream Command Interrupt is generated whenever there is a Data Stream Command in the COMMAND IN SFR. The interrupt is generated to ensure that th internal interrupt is recognized before another instruction is executed.

One instruction from the main program is executed between two consecutive interrupt service routines as in the 80C51. However, if the second interrupt service routine is due to a Data Stream Command Interrupt, the main program instruction is not executed (to prevent misreading of invalid data).

Interrupt Enabling and Priority

Each of the three interrupt special function registers (IE, IP and IEP) is listed below with its corresponding bit definitions.

Interrupt Enable Register (IEC)

Symbolic Address									Physical Address
IEC	EA	—	—	ES	ET1	EX1	ET0	EX0	0A8H
	(MSB)							(LSB)	

Symbol	Position	Function
EA	IE.7	Enables all interrupts. If EA = 0, no interrupt will be acknowledged. If EA = 1, each interrupt source is individually enabled or disabled by setting or clearing its enable bit.
—	IE.6	(reserved)
—	IE.5	(reserved)
ES	IE.4	Serial Channel interrupt enable
ET1	IE.3	Internal Timer/Counter 1 Overflow Interrupt
EX1	IE.2	External Interrupt Request 1.
ET0	IE.1	Internal Timer/Counter 0 Overflow Interrupt
EX0	IE.0	External Interrupt Request 0.

Interrupt Priority Register (IPC)

A priority level of 0 or 1 may be assigned to each interrupt source, with 1 being higher priority level, through the IPC and the IEP (Interrupt Enable and Priority) SFR. A priority level of 1 interrupt can interrupt a priority level 0 service routine to allow nesting of interrupts.

Symbolic Address									Physical Address
IPC	—	—	—	PSC	PT1	PX1	PT0	PX0	0B8H
	(MSB)							(LSB)	

Symbol	Position	Function	Priority Within A Level
			(lowest)
—	IP.7	(reserved)	—
—	IP.6	(reserved)	—
—	IP.5	(reserved)	—
PSC	IP.4	Local Serial Channel	0.7
PT1	IP.3	Internal Timer/Counter 1	0.5
PX1	IP.2	External Interrupt Request 1	0.3
P0	IP.1	Internal Timer/Counter 0	0.1
PX0	IP.0	External Interrupt Request 0	0.0
			(highest)

Interrupt Enable and Priority Register (IEP)

The Interrupt Enable and Priority Register establishes the enabling and priority of those resources not covered in the Interrupt Enable and Interrupt Priority SFRs.

Symbolic Address									Physical Address
IEP	—	—	PFIFO	EDMA0	EDMA1	PDMA0	PDMA1	EFIFO	0F8H
	(MSB)							(LSB)	

Symbol	Position	Function	Priority Within a Level
—	IEP.7	(reserved)	
—	IEP.6	(reserved)	
PFIFO	IEP.5	Slave Bus Interrupt Priority	0.6
EDMA0	IEP.4	DMA Channel 0 Interrupt Enable	
EDMA1	IEP.3	DMA Channel 1 Interrupt Enable	
PDMA0	IEP.2	DMA Channel 0 Priority	0.2
PDMA1	IEP.1	DMA Channel 1 Priority	0.4
EFIFO	IEP.0	Enable FIFO Buffer Interrupt Enable	

FIFO-EXTERNAL HOST INTERFACE FREEZE MODE

Overview

During Freeze Mode the internal CPU can reconfigure the FIFO interface. Freeze Mode is provided to prevent the host from accessing the FIFO during a reconfiguration sequence. The internal CPU invokes Freeze Mode by clearing bit 3 of the Slave Control SFR (SC3). INTRQ becomes active whenever Freeze Mode is invoked to indicate the freeze status. The interrupt can only be deactivated by the Host reading the Host Status SFR.

During Freeze Mode only two operations are possible by the Host to the UPI-452 slave, the balance are disabled, as shown in Table 8. The internal DMA disabled during Freeze Mode, and the internal CPU has write access to all of the FIFO control SFRs (Table 9).

Initialization

At reset, the FIFO - Host interface is automatically frozen (SC3=0). The CBP SFR and the Output FIFO Read and Write Pointers are set to 40H. The Input FIFO Threshold SFR is set to 80H and the Output FIFO Threshold SFR is set to one. The Input FIFO Read and Write Pointer SFRs are set to zero. The Input and Output FIFO channels may be reconfigured by programming any of these Special Function Registers. Once the FIFO channel configuration sequence is complete, the internal CPU should set SC3 (CS3=1) to enable normal FIFO operation.

Table 8. Slave Bus Interface Status During Freeze Mode

Interface Pins; $\overline{DACK}$	$\overline{CS}$	A2	A1	A0	$\overline{READ}$	$\overline{WRITE}$	Operation In Normal Mode	Status In Freeze Mode
1	0	0	1	0	0	1	Read Host Status SFR	Operational
1	0	0	1	1	0	1	Read Host Control SFR	Operational
1	0	0	1	1	1	0	Write Host Control SFR	Disabled
1	0	0	0	0	0	1	Data or DMA Data from Output Channel	Disabled
1	0	0	0	0	1	0	Data or DMA Data to Input Channel	Disabled
1	0	0	0	1	0	1	Data Stream Command from Output Channel	Disabled
1	0	0	0	1	1	0	Data Stream Command to Input Channel	Disabled
1	0	1	0	0	0	1	Read Immediate Command Out from Output Channel	Disabled
1	0	1	0	0	1	0	Write Immediate Command In to Input Channel	Disabled
0	X	X	X	X	0	1	DMA Data from Output Channel	Disabled
0	X	X	X	X	1	0	DMA Data to Input Channel	Disabled

Invoking Freeze Mode During Normal Operation

When the UPI-452 is in normal operation, Freeze Mode should not be arbitrarily invoked by clearing SC3 (SC3 = 0) because the external Host runs asynchronously to the internal CPU. Invoking Freeze Mode without first stopping the external Host from accessing the UPI-452 will not guarantee a clean break with the external Host.

The proper way to invoke Freeze Mode is by issuing an Immediate Command to the external host indicating that Freeze Mode will be invoked. Upon receiving the Immediate Command, the external Host should complete servicing all pending interrupts and DMA requests, then send an Immediate Command back to the UPI-452 acknowledging the Freeze Mode request. After issuing the first Immediate Command, the internal CPU should not perform any action on the FIFO until Freeze Mode is invoked.

If Freeze Mode is invoked without stopping the Host, only the last two bytes of data written into or read from the FIFO will be valid. The timing diagram for disabling the FIFO module to the external Host interface is illustrated in Figure 12. Due to this synchronization sequence, the UPI-452 might not go into Freeze Mode immediately after SC3 is cleared. A special bit in the Slave Status Register (SST5) is provided to indicate the status of the Freeze Mode. The Freeze Mode operations described in this section are only valid after SST5 is cleared.

As Freeze Mode is invoked, the DRQIN or DRQOUT will be deactivated (stopping the transferring of data), bit 1 of the Host Status SFR will be set (HST1 = 1), and SST5 will be cleared (SST5 = 0) to indicate to the external Host and internal CPU that the slave interface has been frozen. After the freeze becomes effective, any attempt by the external Host to access the FIFO will cause the overrun and underrun bits to be activated (bits HST7 (for reads) or HST3 (for writes)). These two bits, HST3 and HST7, will be set (deactivated) after the Host Status SFR has been read.

External Host writing to the Immediate Command In SFR and the Host Control SFR is also inhibited when the slave bus interface is frozen. Writing to these two registers after Freeze Mode is invoked will also cause HST3 (overrun) to be activated (HST3 = 0). Similarly, reading the Immediate Command Out Register by the external Host is disabled during Freeze Mode, and any attempt to do so will cause the clearing (deactivating, "0") of HST7 bit (underrun).

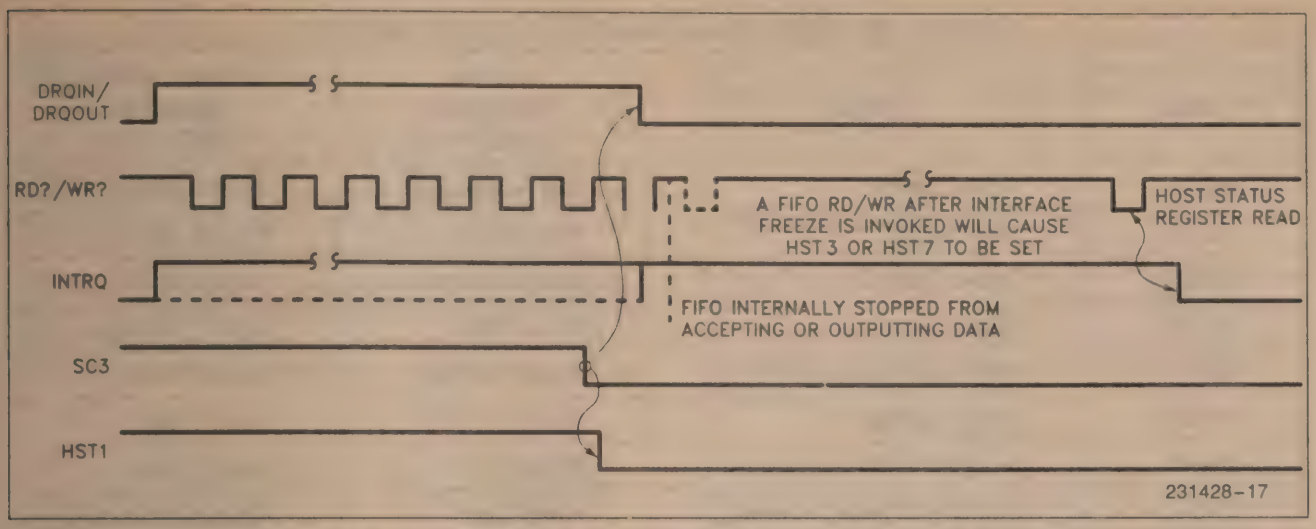


Figure 12. Disabling FIFO to Host Slave Interface Timing Diagram

After the slave bus interface is frozen, the internal CPU can perform the following operations on the FIFO Special Function Registers (these operations are allowed only during Freeze Mode).

For FIFO Reconfiguration	1. Changing the Channel Boundary Pointer SFR. 2. Changing the Input and Output Threshold SFR.
To Enhance the Testability	3. Writing to the read and write pointers of the Input and Output FIFO's. 4. Writing to and reading the Host Control SFRs. 5. Controlling some bits of Host and Slave Status SFRS. 6. Reading the Immediate Command Out SFR and Writing to the Immediate Comand In SFR.

Description of each of these special functions are as follows:

FIFO Module SFRs During Freeze Mode

Table 9 summarizes the characteristics of all the FIFO Special Function Registers during normal and Freeze Modes. The registers that require special treatment in Freeze Mode are: HCON, IWPR, IRPR, OWPR, ORPR, HSTST, SSTAT, IMMIN & IMMOUT SFRs. They can be described in detail as follows:

Host Control SFR (HCON)

During normal operation, this register is written to or read by the external Host. However, in Freeze Mode (i.e. SST5 = 0) the UPI-452 internal CPU has write access to the Host Control SFR and write operations to this SFR by the external Host will not be accepted. If the Host attempts to write to HCON, the Input Channel error condition flag (HST3) will be set.

Input FIFO Pointer Registers (IRPR & IWPR)

Once the FIFO module is in Freeze Mode, error flags due to overrun and underrun of the Input FIFO pointers will be disabled. Any attempt to create an overrun or underrun condition by changing the Input FIFO pointers would result in an inconsistency in performance between the status flag and the threshold counter.

To enhance the speed of the UPI-452, read operations on the Input FIFO will look ahead by two bytes. Hence, every time the IRPR is changed during Freeze Mode, two NOPs need to be executed so that the two byte pipeline can be updated with the new data bytes pointed to by the new IRPR. The

Table 9. FIFO SFR's Characteristics During Freeze Mode

Label	Name	Normal Operation (SST5 = 1)	Freeze Mode Operation (SST5 = 0)
HCON	Host Control	Not Accessible	Read & Write
HSTAT	Host Status	Read Only	Read & Write 4
SLCON	Slave Control	Read & Write	Read & Write
SSTAT	Slave Status	Read Only	Read & Write 4
IEP	Interrupt Enable & Priority	Read & Write	Read & Write
MODE	Mode Register	Read & Write	Read & Write
IWPR	Input FIFO Write Pointer	Read Only	Read & Write 5
IRPR	Input FIFO Read Pointer	Read Only	Read & Write 1, 5
OWPR	Output FIFO Write Pointer	Read Only	Read & Write 6
ORPR	Output FIFO Read Pointer	Read Only	Read & Write 2, 6
CBP	Channel Boundary Pointer	Read Only	Read & Write 3
IMIN	Immediate Command In	Read Only	Read & Write
IMOUT	Immediate Command Out	Read & Write	Read & Write
FIN	FIFO IN	Read Only	Read Only
CIN	COMMAND IN	Read Only	Read Only
FOUT	FIFO OUT	Read & Write	Read & Write
COUT	COMMAND OUT	Read & Write	Read & Write
ITHR	Input FIFO Threshold	Read Only	Read & Write
OTHR	Output FIFO Threshold	Read Only	Read & Write

NOTES:
1. Writing of IRPR will automatically cause the FIFO IN SFR to load the contents of the Input FIFO from that location.
2. Writing to ORPR will automatically cause the IOBL SFR to load the contents of the Output FIFO at that ORPR address.
3. Writing to the CBP SFR will cause automatic reset of the four pointers of the Input and Output FIFO channels.
4. The internal CPU cannot directly change the status of these registers. However, by changing the status of the FIFO channels, the internal CPU can indirectly change the contents of the status registers.
5. Changing the Input FIFO Read/Write Pointers also requires that a consistent update of the Input FIFO Threshold Counter SFR.
6. Changing the Output FIFO Read/Write Pointers also requires that a consistent update of the Output FIFO Threshold Counter SFR.

Threshold Counter SFR also needs to change by the same number of bytes as the IRPR (increase Threshold Counter if IRPR goes forward or decrease if IRPR goes backward). This will ensure that future interrupts will still be generated only after a threshold number of bytes are available. (See "Input and Output FIFO Threshold SFR" section below.)

In Freeze Mode, the internal CPU can also change the content of IWPR, and each change of IWPR also requires an update of the Threshold Counter SFR.

Normally, the internal CPU cannot write into the Input FIFO. It can, however, during Freeze Mode by first reconfiguring the FIFO as an Output FIFO (Refer to "Input and Output FIFO Threshold SFR" section below).Changing the IRPR to be equal to IWPR generates a full condition while changing IWPR to be equal to IRPR generates an empty condition. The order in which the pointers are written determines whether a full or empty condition is generated.

Output FIFO Pointer SFR (ORPR and OWPR)

In Freeze Mode the contents of OWPR can be changed by the internal CPU, but each change of OWPR or ORPR requires the Threshold Counter SFR to be updated as described in the next section. A NOP must be executed whenever a new value is written into ORPR, as just described for changes to IRPR. As before, changing ORPR to be equal to OWPR will generate a full condition, Output FIFO overrun or underrun condition cannot be generated though.

Input and Output FIFO Threshold SFR (ITHR & OTHR)

The Input and Output FIFO Threshold SFRs are also programmable by the internal CPU during Freeze Mode. For proper operation of the Threshold feature, the Threshold SFR should be changed only when the Input and Output FIFO channels are empty, since they reflect the current number of bytes available to read/write before an interrupt is generated.

Table 10 illustrates the Threshold SFRs range of values and the number of bytes to be transferred when the Request For Service Flag is activated:

Table 10. Threshold SFRs Range of Values and Number of Bytes to be Transferred

ITHR (lower seven bits)	No. of Bytes Available to be Written	OTHR (lower seven bits	No. of Bytes Available to be Read
0	CBPR	1	2
1	CBPR-1	2	3
2	CBPR-2	3	4
•	•	•	•
•	•	•	•
•	•	•	•
CBP-3	3	(80H-CBP)-3	(80H-CBP)-2
CBP-2	2	(80H-CBP)-2	(80H-CBP)-1
		(80H-CBP)-1	(80H-CBP)

The eighth bit of the Input and Output FIFO Threshold SFR indicates the status of the service requests regardless of the freeze condition. If the eighth bit is a "1", the FIFO is requesting service from the external Host. In other words, when the Threshold SFR value goes below zero (2's complement), a service request is generated. Normally the ITHR SFR is incremented for each read operation by the Host and decremented for each write by the internal CPU. The OTHR SFR is decremented by internal CPU writes and incremented by external Host reads. Thus if the pointers are moved when the FIFO's are not empty, these relationships can be used to calculate the offset for the Threshold SFRs. It is best to change the Threshold SFRs only when the FIFO's are empty to avoid this complication.

Host Status SFR (HSTAT)

When in Freeze Mode, some bits in the Host Status SFR are forced high and will not reflect the new status until the system returns to normal operation. The definition of the register in Freeze Mode is as follows:

NOTE:
The internal CPU reads this shadow latch value when reading the Host Status SFR. The shadow latch will keep the information for these bits so normal operation can be resumed with the right status. The following bits are cleared (=0) when Freeze Mode is invoked;

HST7 Output FIFO Error Condition Flag

1 = No error.

0 = An invalid read has been done on the output FIFO or the Immediate Command Out Register by the host CPU.

NOTE:
The normal underrun error condition status is disabled. If an Immediate Command Out (IMOUT) SFR read is attempted during Freeze Mode, the ocntents of the IMOUT SFR is output on the Data Buffer and the error status is set (= 1).

HST6 Immediate Command Out SFR Status

During normal operation, this bit is cleared (=0) when the IMOUT SFR is written by the UPI-452 internal CPU and set (= 1) when the IMOUT SFR is read by the external Host. Once the host-slave interface is frozen (i.e. SST5 = 0), this bit will be read as a 1 by the host CPU. A shadow latch will keep the information for this bit so normal operation can be resumed with the correct status.

Shadow latch:

1 = Internal CPU reads the IMOUT SFR

0 = Internal CPU writes to the IMOUT SFR

HST5 Data Stream Command at Output FIFO

This bit is forced to a "1" during Freeze Mode to prevent the external host CPU from trying to read the DSC. Once normal operation is resumed, HST5 will reflect the Data/Command status of the current byte in the Output FIFO.

Shadow Latch (read by the internal CPU):

1 = No Data Stream Command (DSC)

0 = Data Stream Command at Output FIFO

HST4 Output FIFO Service Request Status

When Freeze Mode is invoked, this bit no longer reflects the Output FIFO Request Service Status. This bit wll be forced to a "1".

HST3 Input FIFO Error Condition Flag

1 = No error.

0 = One of the following operations has been attempted by the external host and is invalid:

1) Write into the Input FIFO
2) Write into the Host Control SFR
3) Write into the Immediate Comamnd In SFR

NOTE:
The normal Input FIFO overrun condition is disabled.

HST2 Immediate Command In SFR Status

This bit is normally cleared when the internal CPU reads the IMIN SFR and set when the external host CPU writes into the IMIN SFR. When the host-slave interface is frozen, reading and writing of the IMIN will change the shadow latch of this bit. This bit will be read as a "1" by the external Host.

Shadow latch.

1 = Internal CPU writes into IMIN SFR

0 = Internal CPU reads the IMIN SFR

HST1 Freeze Mode Status

1 = Freeze Mode.

0 = Normal Operation (non-Freeze Mode).

NOTE:
This bit is used to indicate to the external Host that the host-slave interface has been frozen and hence the external Host functions are now reduced as shown in Table 8.

HST0 Input FIFO Request Service Satus

When slave interface is frozen this bit no longer reflects the Input FIFO Request Service Status. This bit will be forced to a "1".

Slave Status SFR (SSTAT)

The Slave Status SFR is a read-only SFR. However, once the slave interface is frozen, most of the bits of this SFR can be changed by the internal CPU by reconfiguring the FIFO and accessing the FIFO Special Function Registers.

SST7 Output FIFO Overrun Error Flag

Inoperative in Freeze Mode.

SST6 Immediate Command Out SFR Status

In Freeze Mode, this bit will be cleared when the internal CPU reads the Immediate Command Out SFR and set when the internal CPU writes to the Immediate Command Out Register.

SST5 FIFO-External Interface Freeze Mode Status

This bit indicates to the internal CPU that Freeze Mode is in progress and that it has write access to the FIFO Control, Host control and Immediate Command SFRs.

SST4 Output FIFO Request Service Status

During normal operation, this bit indicates to the internal CPU that the Output FIFO is ready for more data. The status of this bit reflects the position of the Output FIFO read and write pointers. Hence, in Freeze Mode, this flag can be changed by the internal CPU indirectly as the read and write pointers change.

SST3 Input FIFO Underrun Flag

Inoperative during Freeze Mode.

During normal operation, a read operation clears (=0) this bit when there are no data bytes in the Input FIFO and deactivated (=1) when the Slave Status SFR is read. In Freeze Mode, this bit will not be cleared by an Input FIFO read underrun error condition, nor will it be reset by the reading of the Slave Status SFR.

SST2 Immediate Command In SFR Status

This bit is normally activated (=0) when the external host CPU writes into the Immediate Command In SFR and deactivated (=1) when it is read by the internal CPU. In Freeze Mode, this bit will not be activated (=0) by the external Host's writing of the Immediate Command IN SFR since this function is disabled. However, this bit will be cleared (=0) if the internal CPU writes to the Immediate Command In SFR and it will be set =1) if it reads from the register.

SST1 Data Stream Command at Input FIFO Flag

In Freeze Mode, this bit operates normally. It indicates whether the next byte of data from the Input FIFO is a DSC or data byte. If it is a DSC byte, reading from FIFO IN SFR will result in reading invalid data (FFH) and vice versa. In Freeze Mode, this bit still reflects the type of data byte available from the Input FIFO.

SST0 Input FIFO Service Request Flag

During normal operation, this bit is activated (=0) when the Input FIFO contains bytes that can be read by the internal CPU and deactivated (=1) when the Input FIFO does not need any service from the internal CPU. In Freeze Mode, the status of this bit should not change unless the pointers of the Input FIFO are changed. In this mode, the internal CPU can indirectly change this bit by changing the read and write pointers of the Input FIFO but cannot change it directly.

Immediate Command In/Out SFR (IMMIN/IMMOUT)

If Freeze Mode is in progress, writing to the Immediate Command In SFR by the external host will be disabled, and any such attempt will cause HST3 to be cleared (=0). Similarly, the Immediate Command Out SFR read operation (by the host) will be disabled internally and read attempts will cause HST7 to be cleared (=0).

Internal CPU Read and Write of the FIFO During Freeze Mode

In normal operation, the Input FIFO can only be read by the internal CPU and similarly, the Output FIFO can only be written by the internal CPU. During Freeze Mode, the internal CPU can read the entire contents of the Input FIFO by programming the CBP SFR to 7FH, setting the IRPR SFR to zero, and then the IWPR SFR to zero. Programming the pointer registers in this order generates a FIFO full signal to the FIFO logic and enables internal CPU read operations. If the IWPR and IRPR are already zero, the write pointer should be changed to a non-zero value to clear the empty status then the pointers can be set to zero.

In a similar manner, the internal CPU can write to all 128 bytes of the FIFO by setting the CBP SFR to zero, setting OWPR SFR to zero, and then setting ORPR SFR to zero. This generates a FIFO empty signal and allows internal CPU write operations to all 128 bytes of the FIFO. The Threshold registers also need to be adjusted when the pointers are changed.(See "Input and Output FIFO Threshold SFR" section below.)

MEMORY ORGANIZATION

The UPI-452 has separate address spaces for Program Memory and Data Memory like the 80C51. The Program Memory can be up to 64K bytes. The lower 8K of Program Memory may reside on-chip. The Data Memory consists of 256 bytes of on-chip RAM, up to 64K bytes of off-chip RAM and a number of "SFRs" (Special Function Registers) which appear as yet another set of unique memory addresses. The 80C51 Special Function Registers are listed in Table 11a, and the additional UPI-452 SFRs are listed in Table 11b. A brief description of the 80C51 core SFRs is also provided below.

Accessing External Memory

As in the 80C51, accesses to external memory are of two types: Accesses to external Program Memory and accesses to external Data Memory.

External Program Memory is accessed under two conditions:

1) Whenever signal EA is active; or
2) Whenever the program counter (PC) contains a number that is larger than 1FFFH.

This requires that the ROMless versions have EA wired low to enable the lower 8K program bytes to be fetched from external memory.

External Data Memory is accessed using either the MOVX @DPTR (16 bit address) or the MOVX @Ri (8 bit address) instructions.

Table 11a. 80C51 Special Function Registers;

Symbol	Name	Address
*ACC	Accumulator	0E0H
*B	B Register	0F0H
*PSW	Program Status Word	0D0H
SP	Stack Pointer	81H
DPTR	Data Pointer (consisting of DPH and DPL)	82H
*P0	Port 0	80H
*P1	Port 1	90H
*P2	Port 2	0A0H
*P3	Port 3	0B0H
*IP	Interrupt Priority Control	0B8H
*IE	Interrupt Enable Control	0A8H
TMOD	Timer/Counter Mode Control	89H
TCON	Timer/Counter 2 Control	0C8H
THO	Timer/Counter 0 (high byte)	8CH
TLO	Timer/Counter 0 (low byte)	8AH
TH1	Timer/Counter 1 (high byte)	8DH
TL1	Timer/Counter 1 (low byte)	8BH
*SCON	Serial Control	98H
SBUF	Serial Data Buff	99H
*PCON	Power Control	87H

Table 11b. UPI-452 Additional Special Function Registers

ITHR	Input FIFO Threshold	0F6H
OTHR	Output FIFO Threshold	0F7H
*SLCON	Slave Control	0E8H
SSTAT	Slave Status	0E9H
*IEP	Interrupt Enable & Priority	0F8H
MODE	Mode Register	0F9H
IWPR	Input Write Pointer	0EAH
IRPR	Input Read Pointer	0EBH
ORPR	Output Read Pointer	0FAH
OWPR	Output Write Pointer	0FBH
CBP	Channel Boundary Pointer	0ECH
IMMIN	Immediate Command In	0FCH
IMMOUT	Immediate Command Out	0FDH
FIN	FIFO IN	0EEH
CIN	COMMAND IN	0EFH
FOUT	FIFO OUT	0FEH
COUT	COMMAND OUT	0FFH
*P4	Port 4	0C0H
HSTAT	Host Status	0E6H
HCON	Host Control	0E7H
DCONO	DMA0 Control	92H
DCON1	DMA1 Control	93H
	DMA Source Address	
SARLO	low byte/	0A2H
SARHO	hi byte/ channel 0	0A3H
SARL1	low byte/	0B2H
SARH1	hi byte/ channel 1	0B3H
	DMA Destination Address	
DARLO	low byte/	0C2H
DARHO	hi byte/ channel 0	0C3H
DARL1	low byte/	0D2H
DARH1	hi byte/ channel 1	0D3H
	DMA Byte Count	
BCRLO	low byte/	0E2H
BCRHO	hi byte/ channel 0	0E3H
BCRL1	low byte/	0F2H
BCRH1	hi byte/ channel 1	0F3H

The SFRs marked with an asterisk (*) are both bit- and byte- addressable. The functions of the SFRs are as follows:

Miscellaneous Special Function Register Description

80C51 SFRs

ACCUMULATOR

ACC is the Accumuator SFR. The mnemonics for accumulator-specific instructions, however, refer to the accumulator simply as A.

B REGISTER

The B SFR is used during multiply and divide operations. For other instructions it can be treated as another scratch pad regster.

PROGRAM STATUS WORD

The PSW SFR contains program status information as detailed in Table 12.

STACK POINTER

The Stack Pointer register is 8 bits wide. It is incremented before data is stored during PUSH and CALL executions. While the stack may reside anywhere in on-chip RAM, the Stack Pointer is initialized to 07H after a reset. This causes the stack to begin at location 08H.

DATA POINTER

The Data Pointer (DPTR) consists of a high byte (DPH) and a low byte (DPL). Its intended function is to hold a 16-bit address. It may be manipulated as a 16-bit register or as two independent 8-bit registers.

PORTS 0 TO 4

P0, P1, P2, P3 and P4 are the SFR latches of Ports 0, 1, 2, 3 and 4, respectively.

SERIAL DATA BUFFER

The Serial Data Buffer is actually two separate registers, a transmit buffer and a receive buffer register. When data is moved to SBUF, it goes to the transmit buffer where it is held for serial transmission. (Moving a byte to SBUF is what initiates the transmission.) When data is moved from SBUF, it comes from the receive buffer.

TIMER/COUNTER SFR

Register pairs (TH0, TL0), and (TH1, TL1) are the 16-bit counting registers for Timer/Counters 0 and 2.

POWER CONTROL SFR (PCON)

The PCON Register (Table 13) controls the power down and idle modes in the UPI-452, as well as providing the ability to double the Serial Channel baud rate. There are also two general purpose flag bits available to the user. Bits 5 and 6 are used to set the DMA mode (see "General Purpose DMA Channels" section), and bit 4 is not used.

Table 12. Program Status Word

Symbolic Address									Physical Address
PSW	CY	AC	FO	RS1	RS0	OV	—	P	0D0H
	(MSB)							(LSB)	

Symbol	Position	Name
CY	PSW.7	Carry Flag
AC	PSW.6	Auxiliary Carry (For BCD operations)
F0	PSW.5	Flag 0 (user assignable)
RS1	PSW.4	Register Bank Select bit 1*
RS0	PSW.3	Register Bank Select bit 0*
OV	PSW.2	Overflow Flag
—	PSW.1	(reserved)
P	PSW.0	Parity Flag

*(RS1, RS0) enable internal RAM register banks as follows:

RS1	RS0	Internal RAM Register Bank
0	0	Bank 0
0	1	Bank 1
1	0	Bank 2
1	1	Bank 4

Table 13. PCON Special Function Register

Symbolic Address									Physical Address
PCON	SMOD	ARB	REQ	—	GF1	GF0	PD	IDL	087H
	(MSB)							(LSB)	

Symbol	Position	Function
SMOD	PCON7	Double Baud rate bit. When set to a 1, the baud rate is doubled when the serial port is being used in either Mode 1, 2 or 3.
ARB	PCON6	DMA Arbiter control bit *
REQ	PCON5	DMA Requestor control bit *
—	PCON4	(reserved)
GF1	PCON3	General-purpose flag bit
GF0	PCON2	General-purpose flag bit
PD	PCON1	Power Down bit. Setting this bit activates power down operation.
IDL	PCON0	Idle Mode bit. Setting this bit activates idle mode operation.

*See "DMA Transfer Mode" description.

NOTE:
If 1's are written to PD and IDL at the same time, PD takes precedence. The reset value of PCON is (000X0000).

intel®

UPI-41A™

PRELIMINARY

UNIVERSAL PERIPHERAL INTERFACE 8-BIT SLAVE MICROCONTROLLER

- **8-Bit CPU plus ROM, RAM, I/O, Timer and Clock in a Single Package**
- **One 8-Bit Status and Two Data Registers for Asynchronous Slave-to-Master Interface**
- **DMA, Interrupt, or Polled Operation Supported**
- **1024 x 8 ROM/EPROM, 64 x 8 RAM, 8-Bit Timer/Counter, 18 Programmable I/O Pins**
- **3.6 MHz 8741A-8 Available**
- **Fully Compatible With All Microprocessor Families**
- **Interchangeable ROM and EPROM Versions**
- **Expandable I/O**
- **RAM Power-Down Capability**
- **Over 90 Instructions: 70% Single Byte**
- **Available in EXPRESS**
  —Standard Temperature Range
  —Extended Temperature Range
- **8741A Available in 40-Lead Cerdip Package. 8041A in Both 40-Lead Plastic and 44-Lead Plastic Leaded Chip Carrier Packages.**
  (See Packaging Specifications, Order #231369)

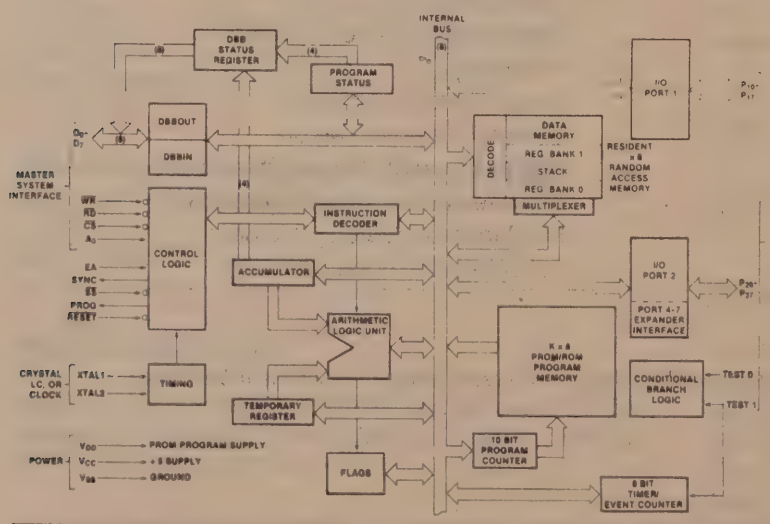
The Intel® UPI-41A™ is a general purpose, programmable interface device designed for use with a variety of 8-bit microprocessor systems. It contains a low cost microcomputer with program memory, data memory, 8-bit CPU, I/O ports, timer/counter, and clock in a single 40-pin package. Interface registers are included to enable the UPI device to function as a peripheral controller in MCS-48™, MCS-80™, MCS-85™, MCS-86™, and other 8-bit systems.

The UPI-41A™ has 1K words of program memory and 64 words of data memory on-chip. To allow full user flexibility the program memory is available as ROM in the 8041A version or as UV-erasable EPROM in the 8741A version. The 8741A and the 8041A are fully pin compatible for easy transition from prototype to production level designs.

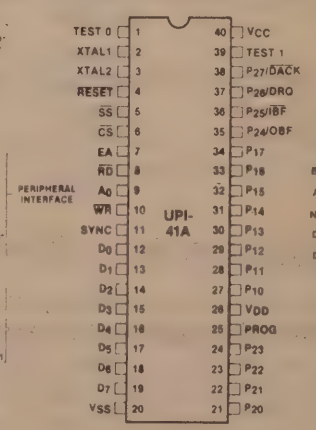
The device has two 8-bit, TTL compatible I/O ports and two test inputs. Individual port lines can function as either inputs or outputs under software control. I/O can be expanded with the 8243 device which is directly compatible and has 16 I/O lines. An 8-bit programmable timer/counter is included in the UPI device for generating timing sequences or counting external inputs. Additional UPI features include: single 5V supply, low power standby mode (in the 8041A), single-step mode for debug (in the 8741A), and dual working register banks.

Because it's a complete microcomputer, the UPI provides more flexibility for the designer than conventional LSI interface devices. It is designed to be an efficient controller as well as an arithmetic processor. Applications include keyboard scanning, printer control, display multiplexing and similar functions which involve interfacing peripheral devices to microprocessor systems.

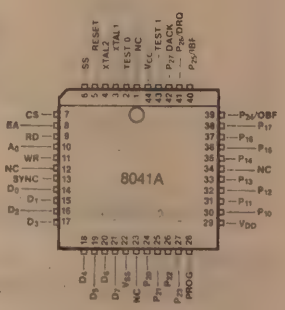
BLOCK DIAGRAM



DIP PIN CONFIGURATION



PLCC PIN CONFIGURATION



Intel Corporation Assumes No Responsibility for the Use of Any Circuitry Other Than Circuitry Embodied in an Intel Product. No Other Circuit Patent Licenses are Implied.
©INTEL CORPORATION, 1985

SEPT. 1985
231316-002

Table 1. Pin Description

Signal	Description
D_0–D_7 (BUS)	Three-state, bidirectional DATA BUS BUFFER lines used to interface the UPI-41A to an 8-bit master system data bus.
P_{10}–P_{17}	8-bit, PORT 1 quasi-bidirectional I/O lines.
P_{20}–P_{27}	8-bit, PORT 2 quasi-bidirectional I/O lines. The lower 4 bits (P_{20}–P_{23}) interface directly to the 8243 I/O expander device and contain address and data information during PORT 4–7 access. The upper 4 bits (P_{24}–P_{27}) can be programmed to provide Interrupt Request and DMA Handshake capability. Software control can configure P_{24} as OBF (Output Buffer Full), P_{25} as $\overline{IBF}$ (Input Buffer Full), P_{26} as DRQ (DMA Request), and P_{27} as $\overline{DACK}$ (DMA ACKnowledge).
$\overline{WR}$	I/O write input which enables the master CPU to write data and command words to the UPI-41A INPUT DATA BUS BUFFER.
$\overline{RD}$	I/O read input which enables the master CPU to read data and status words from the OUTPUT DATA BUS BUFFER or status register.
$\overline{CS}$	Chip select input used to select one UPI-41A out of several connected to a common data bus.
A_0	Address input used by the master processor to indicate whether byte transfer is data or command. During a write operation flag F_1 is set to the status of the A_0 input.
TEST 0, TEST 1	Input pins which can be directly tested using conditional branch instructions. T_1 also functions as the event timer input (under software control). T_0 is used during PROM programming and verification in the 8741A.

Signal	Description
XTAL1, XTAL2	Inputs for a crystal, LC or an external timing signal to determine the internal oscillator frequency.
SYNC	Output signal which occurs once per UPI-41A instruction cycle. SYNC can be used as a strobe for external circuitry; it is also used to synchronize single step operation.
EA	External access input which allows emulation, testing and PROM/ROM verification.
PROG	Multifunction pin used as the program pulse input during PROM programming. During I/O expander access the PROG pin acts as an address/data strobe to the 8243.
$\overline{RESET}$	Input used to reset status flip-flops and to set the program counter to zero. $\overline{RESET}$ is also used during PROM programming and verification. $\overline{RESET}$ should be held low for a minimum of 8 instruction cycles after power-up.
$\overline{SS}$	Single step input used in the 8741A in conjunction with the SYNC output to step the program through each instruction.
V_{CC}	+5V main power supply pin.
V_{DD}	+5V during normal operation. +25V during programming operation. Low power standby pin in ROM version.
V_{SS}	Circuit ground potential.

PROGRAMMING, VERIFYING, AND ERASING THE 8741A EPROM

Programming Verification

In brief, the programming process consists of: activating the program mode, applying an address, latching the address, applying data, and applying a programming pulse. Each word is programmed completely before moving on to the next and is followed by a verification step. The following is a list of the pins used for programming and a description of their functions:

Pin	Function
XTAL 1	Clock Input (1 to 6MHz)
$\overline{\text{Reset}}$	Initialization and Address Latching
Test 0	Selection of Program or Verify Mode
EA	Activation of Program/Verify Modes
BUS	Address and Data Input Data Output During Verify
P20-1	Address Input
V_{DD}	Programming Power Supply
PROG	Program Pulse Input

WARNING:

An attempt to program a missocketed 8741A will result in severe damage to the part. An indication of a properly socketed part is the appearance of the SYNC clock output. The lack of this clock may be used to disable the programmer.

The Program/Verify sequence is:

1. A_0 = 0V, $\overline{CS}$ = 5V, EA = 5V, $\overline{RESET}$ = 0V, TEST0 = 5V, V_{DD} = 5V, clock applied or internal oscillator operating, BUS and PROG floating.
2. Insert 8741A in programming socket
3. TEST 0 = 0v (select program mode)
4. EA = 23V (activate program mode)
5. Address applied to BUS and P20-1
6. $\overline{RESET}$ = 5v (latch address)
7. Data applied to BUS
8. V_{DD} = 25v (programming power)
9. PROG = 0v followed by one 50ms pulse to 23V
10. V_{DD} = 5v
11. TEST 0 = 5v (verify mode)
12. Read and verify data on BUS
13. TEST 0 = 0v
14. $\overline{RESET}$ = 0v and repeat from step 5
15. Programmer should be at conditions of step 1 when 8741A is removed from socket.

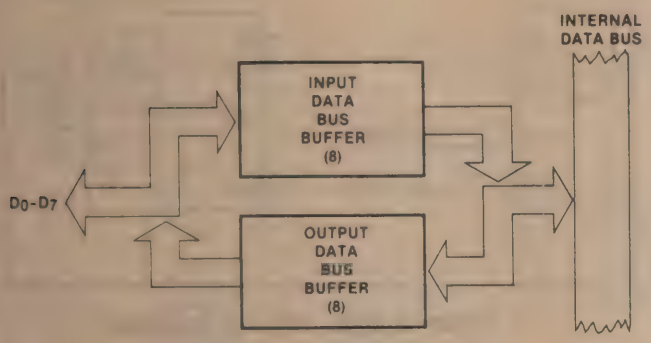
8741A Erasure Characteristics

The erasure characteristics of the 8741A are such that erasure begins to occur when exposed to light with wavelengths shorter than approximately 4000 Angstroms (Å). It should be noted that sunlight and certain types of fluorescent lamps have wavelengths in the 3000–4000Å range. Data show that constant exposure to room level fluorescent lighting could erase the typical 8741A in approximately 3 years while it would take approximately one week to cause erasure when exposed to direct sunlight. If the 8741A is to be exposed to these types of lighting conditions for extended periods of time, opaque labels are available from Intel which should be placed over the 8741A window to prevent unintentional erasure.

The recommended erasure procedure for the 8741A is exposure to shortwave ultraviolet light which has a wavelength of 2537Å. The integrated dose (i.e., UV intensity × exposure time) for erasure should be a minimum of 15 w-sec/cm^2. The erasure time with this dosage is approximately 15 to 20 minutes using an ultraviolet lamp with a 12,000 μW/cm^2 power rating. The 8741A should be placed within one inch of the lamp tubes during erasure. Some lamps have a filter on their tubes which should be removed before erasure.

UPI-41A™ FEATURES AND ENHANCEMENTS

1. Two Data Bus Buffers, one for input and one for output. This allows a much cleaner Master/Slave protocol.



2. 8 Bits of Status

ST_7	ST_6	ST_5	ST_4	F_1	F_0	IBF	OBF
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

ST_4-ST_7 are user definable status bits. These bits are defined by the "MOV STS, A" single byte, single cycle instruction. Bits 4-7 of the accumulator are moved to bits 4-7 of the status register. Bits 0-3 of the status register are not affected.

MOV STS, A Op Code: 90H

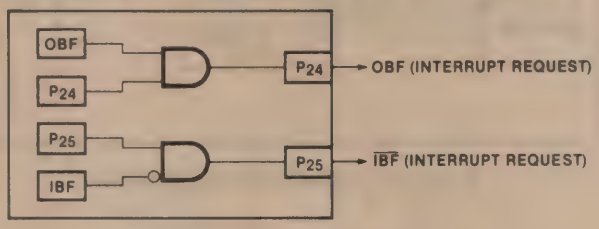
1	0	0	1	0	0	0	0
D_7							D_0

3. $\overline{RD}$ and $\overline{WR}$ are edge triggered. IBF, OBF, F_1 and INT change internally after the trailing edge of $\overline{RD}$ or $\overline{WR}$.

4. P_{24} and P_{25} are port pins or Buffer Flag pins which can be used to interrupt a master processor. These pins default to port pins on Reset.

If the "EN FLAGS" instruction has been executed, P_{24} becomes the OBF (Output Buffer Full) pin. A "1" written to P_{24} enables the OBF pin (the pin outputs the OBF Status Bit). A "0" written to P_{24} disables the OBF pin (the pin remains low). This pin can be used to indicate that valid data is available from the UPI-41A (in Output Data Bus Buffer).

If "EN FLAGS" has been executed, P_{25} becomes the $\overline{IBF}$ (Input Buffer Full) pin. A "1" written to P_{25} enables the $\overline{IBF}$ pin (the pin outputs the inverse of the IBF Status Bit). A "0" written to P_{25} disables the $\overline{IBF}$ pin (the pin remains low). This pin can be used to indicate that the UPI-41A is ready for data.



DATA BUS BUFFER INTERRUPT CAPABILITY

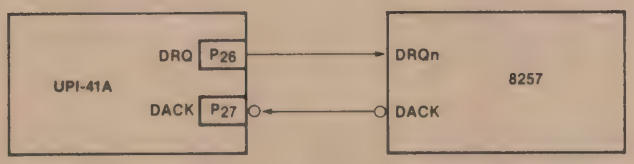
EN FLAGS Op Code: 0F5H

1	1	1	1	0	1	0	1
D_7							D_0

5. P_{26} and P_{27} are port pins or DMA handshake pins for use with a DMA controller. These pins default to port pins on Reset.

If the "EN DMA" instruction has been executed, P_{26} becomes the DRQ (DMA ReQuest) pin. A "1" written to P_{26} causes a DMA request (DRQ is activated). DRQ is deactivated by DACK·RD, DACK·WR, or execution of the "EN DMA" instruction.

If "EN DMA" has been executed, P_{27} becomes the $\overline{DACK}$ (DMA ACKnowledge) pin. This pin acts as a chip select input for the Data Bus Buffer registers during DMA transfers.



DMA HANDSHAKE CAPABILITY

EN DMA Op Code: 0E5H

1	1	1	0	0	1	0	1
D_1							D_0

APPLICATIONS

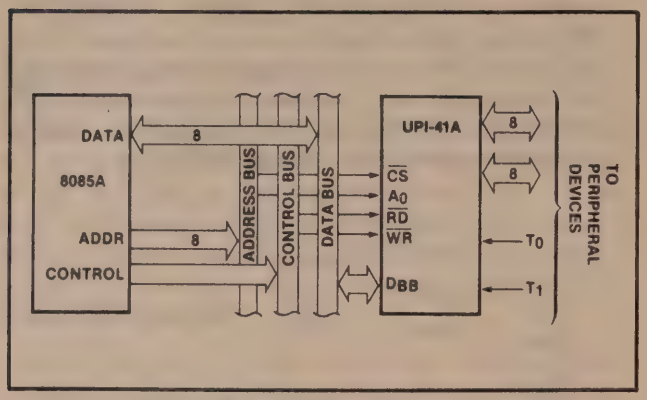


Figure 1. 8085A-UPI-41A Interface

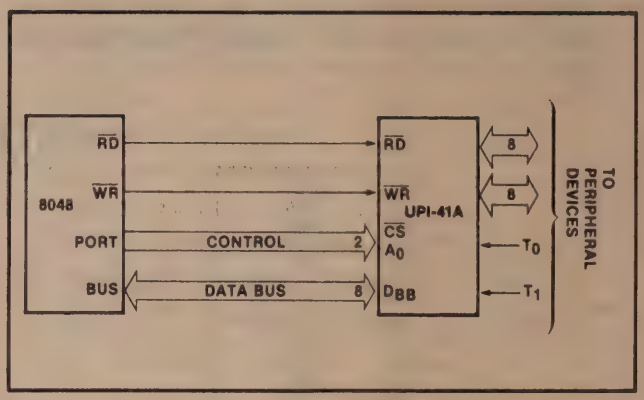


Figure 2. 8048-UPI-41A Interface

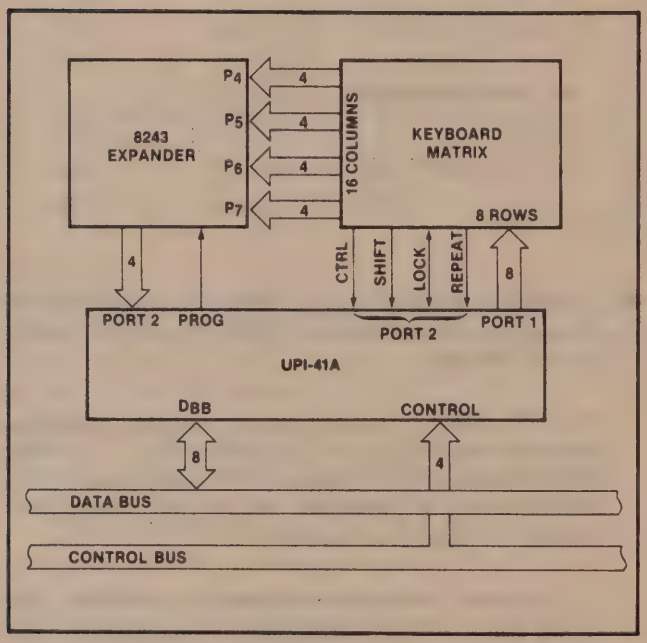


Figure 3. UPI-41A-8243 Keyboard Scanner

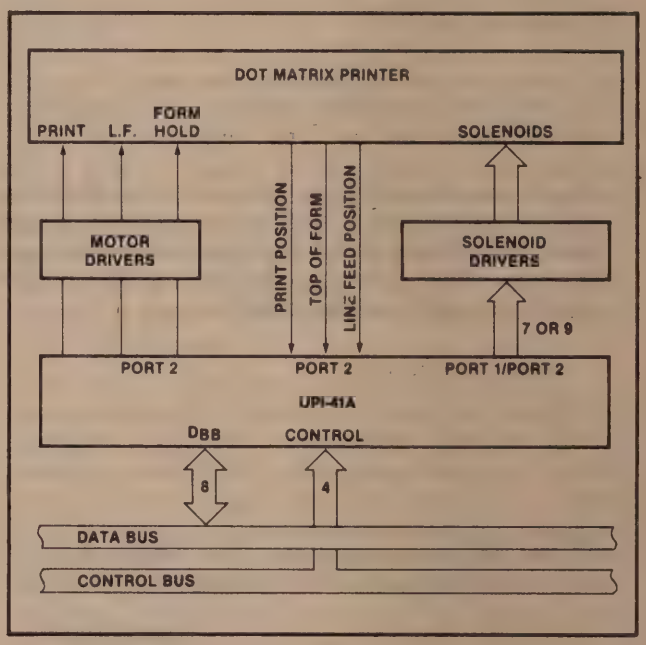


Figure 4. UPI-41A Matrix Printer Interface

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias . . . . . 0°C to 70°C
Storage Temperature . . . . . – 65°C to + 150°C
Voltage on Any Pin With Respect
to Ground . . . . . –0.5V to +7V
Power Dissipation . . . . . 1.5 Watt

*COMMENT: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

D.C. AND OPERATING CHARACTERISTICS

T_A=0°C to 70°C, V_{SS}=0V, V_{CC}=V_{DD}=+5V ±10%*

Symbol	Parameter	Min.	Max.	Unit	Test Conditions
V_{IL}	Input Low Voltage (Except XTAL1, XTAL2, $\overline{RESET}$)	– 0.5	0.8	V	
V_{IL1}	Input Low Voltage (XTAL1, XTAL2, $\overline{RESET}$)	– 0.5	0.6	V	
V_{IH}	Input High Voltage (Except XTAL1, XTAL2, $\overline{RESET}$)	2.2	V_{CC}		
V_{IH1}	Input High Voltage (XTAL1, XTAL2, $\overline{RESET}$)	3.8	V_{CC}	V	
V_{OL}	Output Low Voltage (D_0-D_7)		0.45	V	I_{OL} = 2.0 mA
V_{OL1}	Output Low Voltage ($P_{10}P_{17}$, $P_{20}P_{27}$, Sync)		0.45	V	I_{OL} = 1.6 mA
V_{OL2}	Output Low Voltage (Prog)		0.45	V	I_{OL} = 1.0 mA
V_{OH}	Output High Voltage (D_0-D_7)	2.4		V	I_{OH} = – 400 μA
V_{OH1}	Output High Voltage (All Other Outputs)	2.4		V	I_{OH} = – 50 μA
I_{IL}	Input Leakage Current (T_0, T_1, $\overline{RD}$, $\overline{WR}$, $\overline{CS}$, A_0, EA)		± 10	μA	$V_{SS} \le V_{IN} \le V_{CC}$
I_{OZ}	Output Leakage Current (D_0-D_7, High Z State)		± 10	μA	V_{SS} + 0.45 $\le V_{IN} \le V_{CC}$
I_{LI}	Low Input Load Current ($P_{10}P_{17}$, $P_{20}P_{27}$)		0.5	mA	V_{IL} = 0.8V
I_{LI1}	Low Input Load Current ($\overline{RESET}$, $\overline{SS}$)		0.2	mA	V_{IL} = 0.8V
I_{DD}	V_{DD} Supply Current		15	mA	Typical = 5 mA
I_{CC} + I_{DD}	Total Supply Current		125	mA	Typical = 60 mA

A.C. CHARACTERISTICS

T_A=0°C to 70°C, V_{SS}=0V, V_{CC}=V_{DD}=+5V ±10%*

DBB READ

Symbol	Parameter	Min.	Max.	Unit	Test Conditions
t_{AR}	$\overline{CS}$, A_0 Setup to $\overline{RD}$↓	0		ns	
t_{RA}	$\overline{CS}$, A_0 Hold After $\overline{RD}$↑	0		ns	
t_{RR}	$\overline{RD}$ Pulse Width	250		ns	
t_{AD}	$\overline{CS}$, A_0 to Data Out Delay		225	ns	C_L = 150 pF
t_{RD}	$\overline{RD}$↓ to Data Out Delay		225	ns	C_L = 150 pF
t_{DF}	$\overline{RD}$↑ to Data Float Delay		100	ns	
t_{CY}	Cycle Time (Except 8741A-8)	2.5	15	μs	6.0 MHz XTAL
t_{CY}	Cycle Time (8741A-8)	4.17	15	μs	3.6 MHz XTAL

DBB WRITE

Symbol	Parameter	Min.	Max.	Unit	Test Conditions
t_{AW}	$\overline{CS}$, A_0 Setup to $\overline{WR}$↓	0		ns	
t_{WA}	$\overline{CS}$, A_0 Hold After $\overline{WR}$↑	0		ns	
t_{WW}	$\overline{WR}$ Pulse Width	250		ns	
t_{DW}	Data Setup to $\overline{WR}$↑	150		ns	
t_{WD}	Data Hold After $\overline{WR}$↑	0		ns	

A.C. TIMING SPECIFICATION FOR PROGRAMMING

T_A=0°C to 70°C, V_{CC}=+5V ±10% *

Symbol	Parameter	Min.	Max.	Unit	Test Conditions
t_{AW}	Address Setup Time to $\overline{RESET}$ ↑	4tcy			
t_{WA}	Address Hold Time After $\overline{RESET}$ ↑	4tcy			
t_{DW}	Data in Setup Time to PROG ↑	4tcy			
t_{WD}	Data in Hold Time After PROG ↓	4tcy			
t_{PH}	$\overline{RESET}$ Hold Time to Verify	4tcy			
t_{VDDW}	V_{DD} Setup Time to PROG ↑	4tcy			
t_{VDDH}	V_{DD} Hold Time After PROG ↓	0			
t_{PW}	Program Pulse Width	50	60	mS	
t_{TW}	Test 0 Setup Time for Program Mode	4tcy			
t_{WT}	Test 0 Hold Time After Program Mode	4tcy			
t_{DO}	Test 0 to Data Out Delay		4tcy		
t_{WW}	$\overline{RESET}$ Pulse Width to Latch Address	4tcy			
t_r, t_f	V_{DD} and PROG Rise and Fall Times	0.5	2.0	μs	
t_{CY}	CPU Operation Cycle Time	5.0		μs	
t_{RE}	$\overline{RESET}$ Setup Time Before EA ↑.	4tcy			

Note: If TEST 0 is high, t_{DO} can be triggered by $\overline{RESET}$ ↑.

* For Extended Temperature EXPRESS, use M8741A electrical parameters.

D.C. SPECIFICATION FOR PROGRAMMING

T_A = 25°C ±5°C, V_{CC} = 5V ±5%, V_{DD} = 25V ±1V

Symbol	Parameter	Min.	Max.	Unit	Test Conditions
V_{DOH}	V_{DD} Program Voltage High Level	24.0	26.0	V	
V_{DDL}	V_{DD} Voltage Low Level	4.75	5.25	V	
V_{PH}	PROG Program Voltage High Level	21.5	24.5	V	
V_{PL}	PROG Voltage Low Level		0.2	V	
V_{EAH}	EA Program or Verify Voltage High Level	21.5	24.5	V	
V_{EAL}	EA Voltage Low Level		5.25	V	
I_{DD}	V_{DD} High Voltage Supply Current		30.0	mA	
I_{PROG}	PROG High Voltage Supply Current		16.0	mA	
I_{EA}	EA High Voltage Supply Current		1.0	mA	

A.C. CHARACTERISTICS—PORT 2

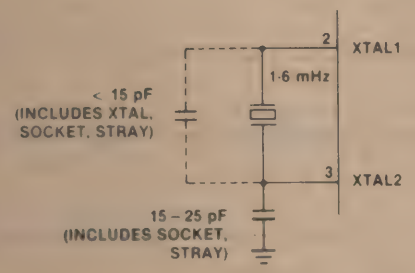
T_A=0°C to 70°C, : V_{CC}= +5V ±10%

Symbol	Parameter	Min.	Max.	Unit	Test Conditions
t_{CP}	Port Control Setup Before Falling Edge of PROG	110		ns	
t_{PC}	Port Control Hold After Falling Edge of PROG	100		ns	
t_{PR}	PROG to Time P2 Input Must Be Valid		810	ns	
t_{PF}	Input Data Hold Time	0	150	ns	
t_{DP}	Output Data Setup Time	250		ns	
t_{PD}	Output Data Hold Time	65		ns	
t_{PP}	PROG Pulse Width	1200		ns	

A.C. CHARACTERISTICS—DMA

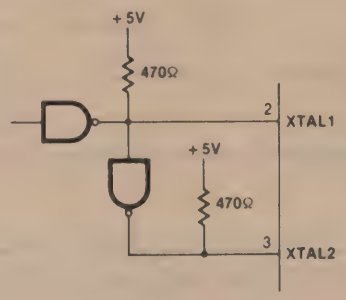
Symbol	Parameter	Min.	Max.	Unit	Test Conditions
t_{ACC}	$\overline{DACK}$ to $\overline{WR}$ or $\overline{RD}$	0		ns	
t_{CAC}	$\overline{RD}$ or $\overline{WR}$ to $\overline{DACK}$	0		ns	
t_{ACD}	$\overline{DACK}$ to Data Valid		225	ns	$C_L = 150$ pF
t_{CRQ}	$\overline{RD}$ or $\overline{WR}$ to DRQ Cleared		200	ns	

CRYSTAL OSCILLATOR MODE



CRYSTAL SERIES RESISTANCE SHOULD BE <75Ω AT 6 MHz; <180Ω AT 3.6 MHz.

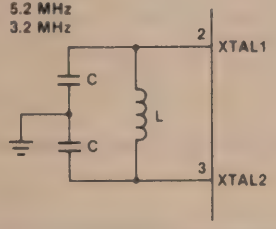
DRIVING FROM EXTERNAL SOURCE



BOTH XTAL1 AND XTAL2 SHOULD BE DRIVEN. RESISTORS TO V_{CC} ARE NEEDED TO ENSURE $V_{IH} = 3.8V$ IF TTL CIRCUITRY IS USED.

LC OSCILLATOR MODE

L	C	NOMINAL f
45 μH	20 pF	5.2 MHz
120 μH	20 pF	3.2 MHz



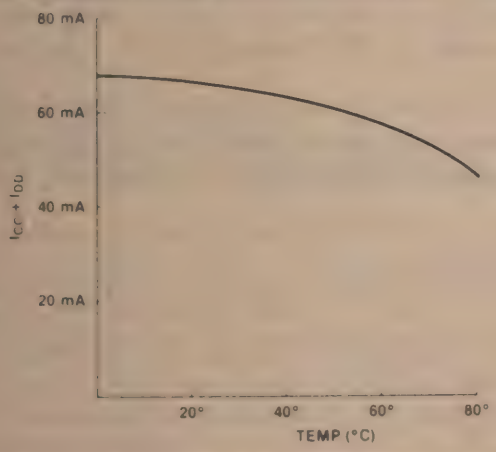
$$f \approx \frac{1}{2\pi\sqrt{LC'}}$$

$$C' = \frac{C + 3C_{PP}}{2}$$

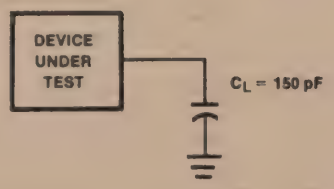
$C_{PP} \approx 5 - 10$ pF PIN-TO-PIN CAPACITANCE

EACH C SHOULD BE APPROXIMATELY 20 pF, INCLUDING STRAY CAPACITANCE.

TYPICAL 8041/8741A CURRENT

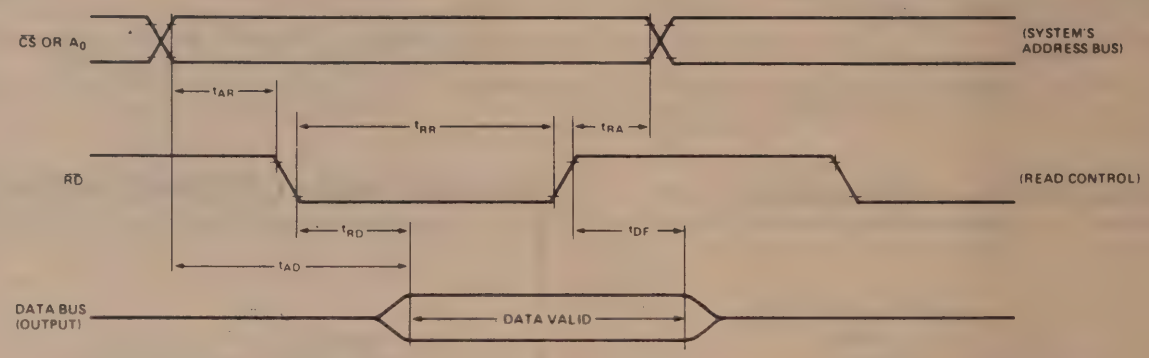


A.C. TESTING LOAD CIRCUIT

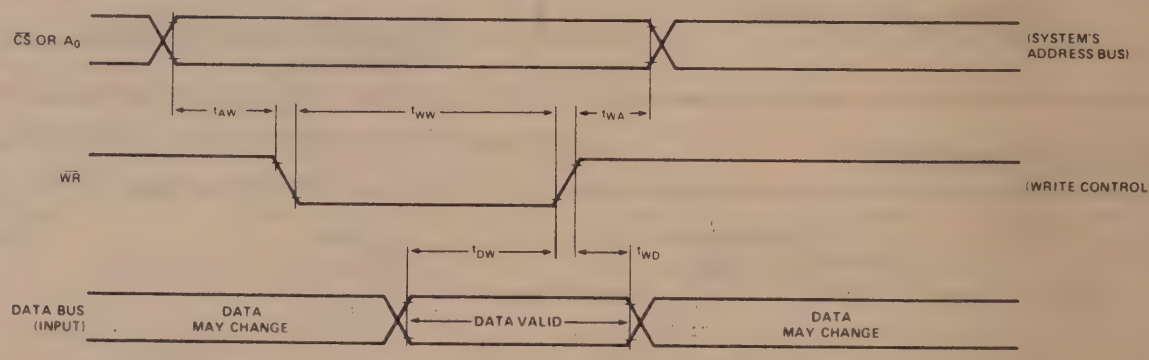


WAVEFORMS

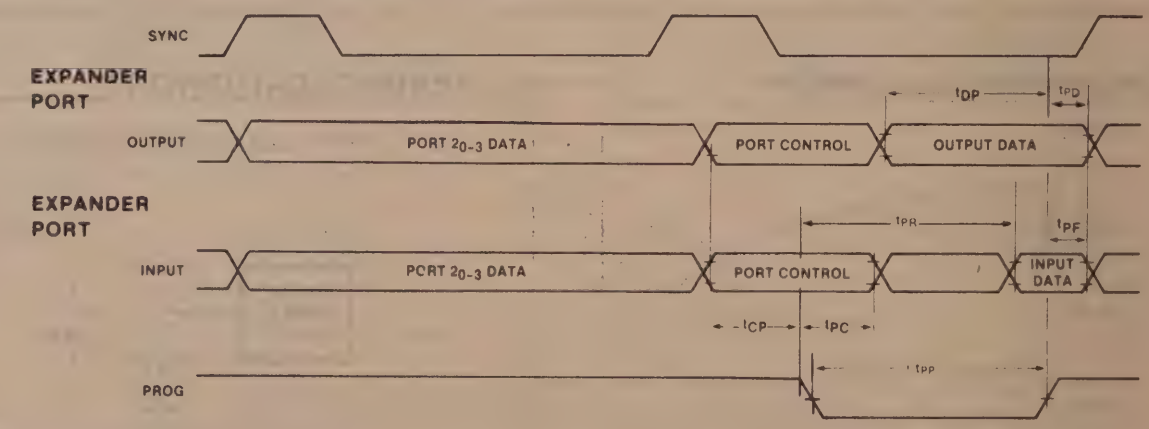
READ OPERATION—DATA BUS BUFFER REGISTER.



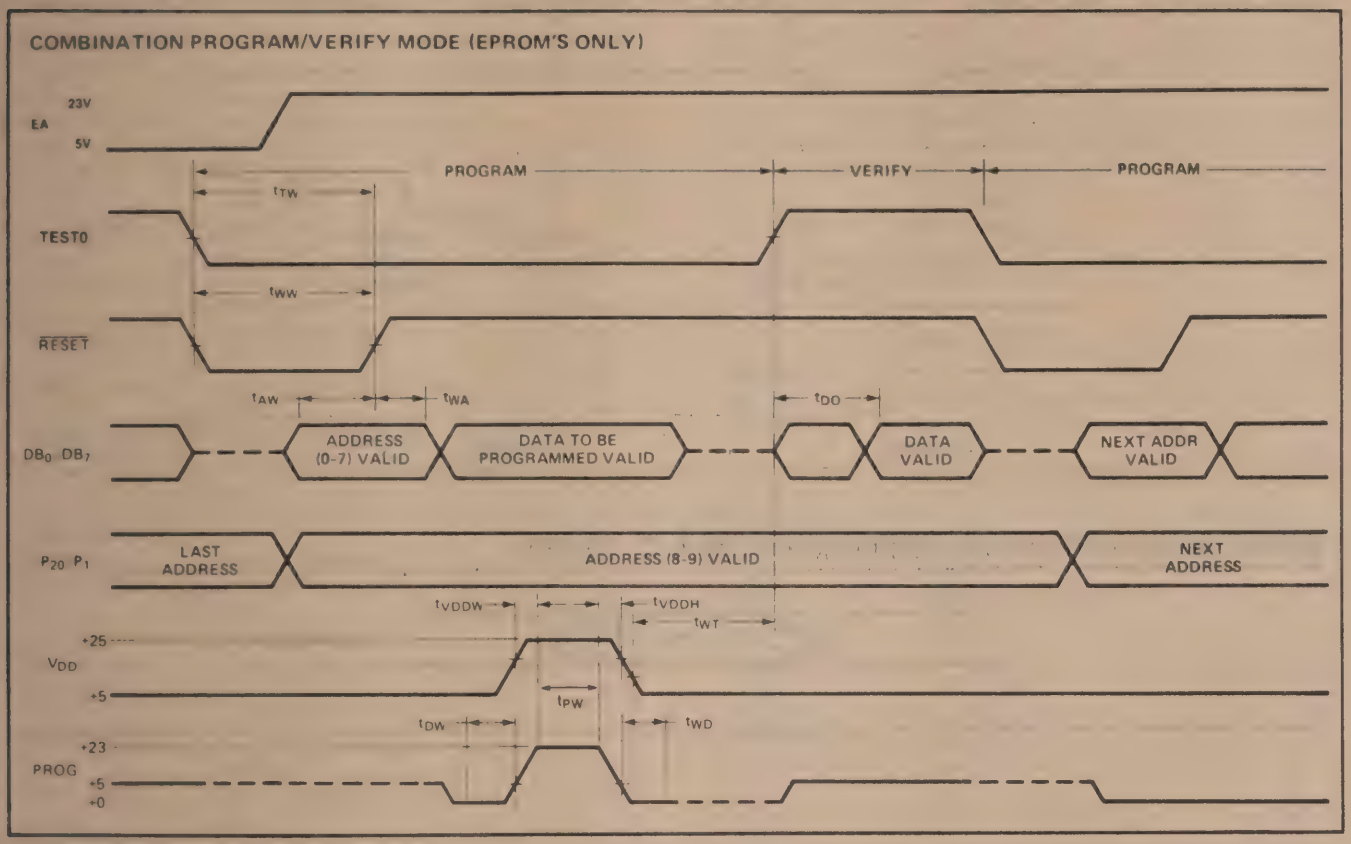
WRITE OPERATION—DATA BUS BUFFER REGISTER.

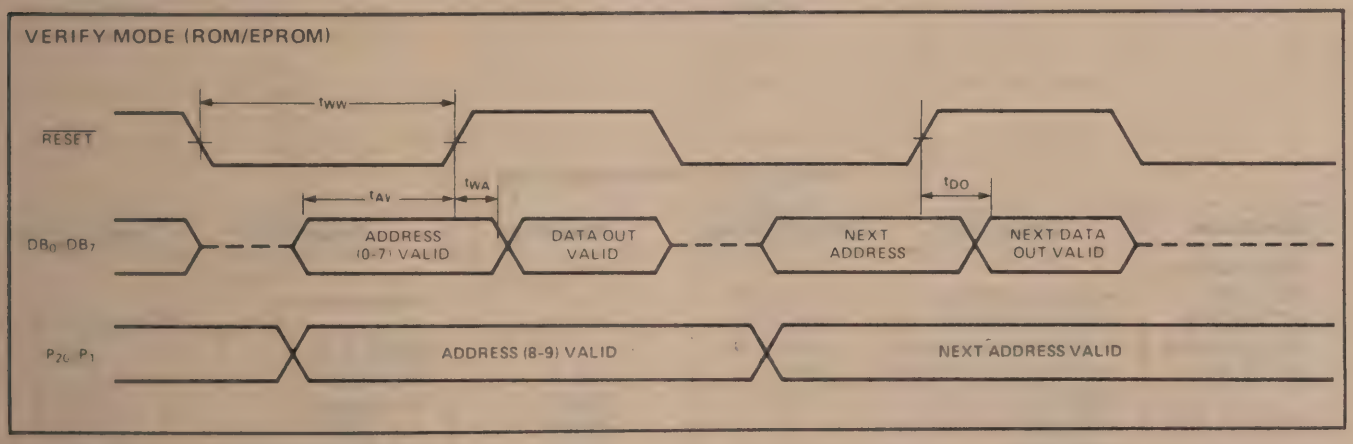


PORT 2 TIMING



WAVEFORMS FOR PROGRAMMING



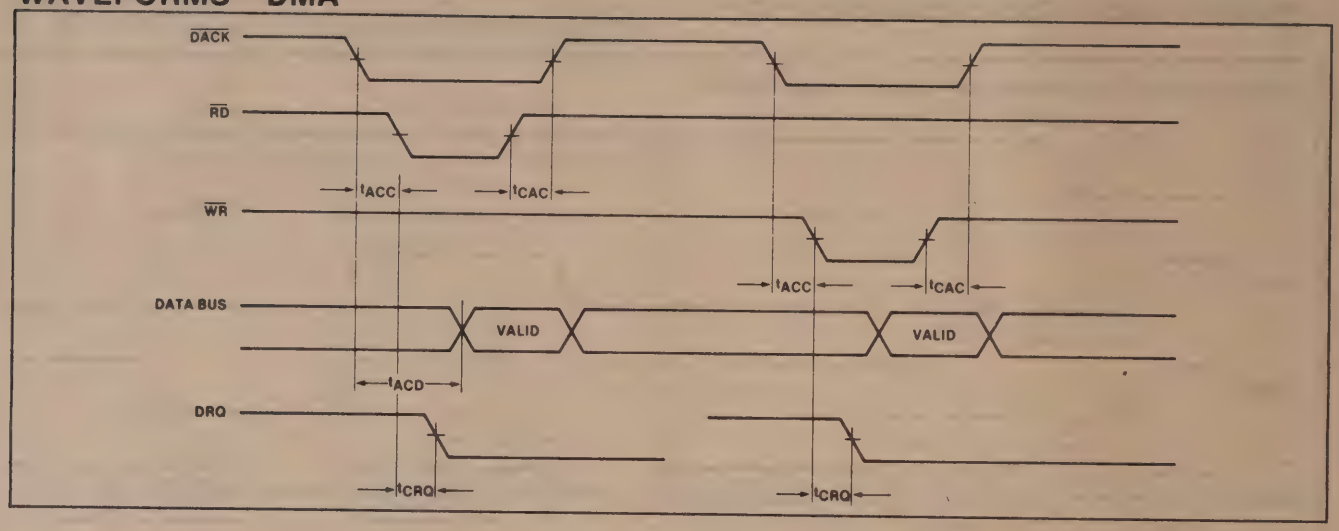


NOTES:
1. PROG MUST FLOAT IF EA IS LOW (i.e., ≠23V), OR IF T0 = 5V FOR THE 8741A. FOR THE 8041A PROG MUST ALWAYS FLOAT.
2. XTAL1 AND XTAL 2 DRIVEN BY 3.6 MHz CLOCK WILL GIVE 4.17 μsec t_{CY}. THIS IS ACCEPTABLE FOR 8741A-8 PARTS AS WELL AS STANDARD PARTS.
3. A0 MUST BE HELD LOW (i.e., = 0V) DURING PROGRAM/VERIFY MODES.

The 8741A EPROM can be programmed by either of two Intel products:

1. PROMPT-48 Microcomputer Design Aid, or
2. Universal PROM Programmer (UPP series) peripheral of the Intellec® Development System with a UPP-848 Personality Card.

WAVEFORMS—DMA



INPUT AND OUTPUT WAVEFORMS FOR A.C. TESTS

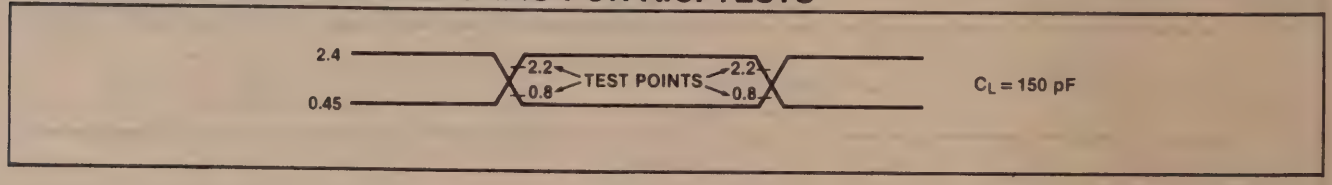


Table 2. UPI™ Instruction Set

Mnemonic	Description	Bytes	Cycles
Accumulator			
ADD A,Rr	Add register to A	1	1
ADD A,@Rr	Add data memory to A	1	1
ADD A,#data	Add immediate to A	2	2
ADDC A,Rr	Add register to A with carry	1	1
ADDC A,@Rr	Add data memory to A with carry	1	1
ADDC A, #data	Add immed. to A with carry	2	2
ANL A,Rr	AND register to A	1	1
ANL A,@Rr	AND data memory to A	1	1
ANL A,#data	AND immediate to A	2	2
ORL A,Rr	OR register to A	1	1
ORL A,@Rr	OR data memory to A	1	1
ORL A,#data	OR immediate to A	2	2
XRL A,Rr	Exclusive OR register to A	1	1

Mnemonic	Description	Bytes	Cycles
XRL A,@Rr	Exclusive OR data memory to A	1	1
XRL A,#data	Exclusive OR immediate to A	2	2
INC A	Increment A	1	1
DEC A	Decrement A	1	1
CLR A	Clear A	1	1
CPL A	Complement A	1	1
DA A	Decimal Adjust A	1	1
SWAP A	Swap nibbles of A	1	1
RL A	Rotate A left	1	1
RCL A	Rotate A left through carry	1	1
RR A	Rotate A right	1	1
RRC A	Rotate A right through carry	1	1

Table 2. UPI™ Instruction Set (Cont'd.)

Mnemonic	Description	Bytes	Cycles
Input/Output			
In A,Pp	Input port to A	1	2
OUTL Pp,A	Output A to port	1	2
ANL Pp,#data	AND immediate to port	2	2
ORL Pp,#data	OR immediate to port	2	2
In A,DBB	Input DBB to A, clear IBF	1	1
OUT DBB,A	Output A to DBB, set OBF	1	1
MOV STS,A	A_4–A_7 to Bits 4–7 of Status	1	1
MOVD A,Pp	Input Expander port to A	1	2
MOVD Pp,A	Output A to Expander port	1	2
ANLD Pp,A	AND A to Expander	1	2
ORLD Pp,A	OR A to Expander port	1	2
Data Moves			
MOV A,Rr	Move register to A	1	1
MOV A,@Rr	Move data memory to A	1	1
MOV A,#data	Move immediate to A	2	2
MOV Rr,A	Move A to register	1	1
MOV @Rr,A	Move A to data memory	1	1
MOV Rr,#data	Move immediate to register	2	2
MOV @Rr, #data	Move immediate to data memory	2	2
MOV A,PSW	Move PSW to A	1	1
MOV PSW,A	Move A to PSW	1	1
XCH A,Rr	Exchange A and register	1	1
XCH A,@Rr	Exchange A and data memory	1	1
XCHD A,@Rr	Exchange digit of A and register	1	1
MOVP A,@A	Move to A from current page	1	2
MOVP3, A,@A	Move to A from page 3	1	2
Timer/Counter			
MOV A,T	Read Timer/Counter	1	1
MOV T,A	Load Timer/Counter	1	1
STRT T	Start Timer	1	1
STRT CNT	Start Counter	1	1
STOP TCNT	Stop Timer/Counter	1	1
EN TCNTI	Enable Timer/Counter	1	1
DIS TCNTI	Disable Timer/ Counter Interrupt	1	1

Mnemonic	Description	Bytes	Cycles
Control			
EN DMA	Enable DMA Hand-shake Lines	1	1
EN I	Enable IBF Interrupt	1	1
DIS I	Disable IBF Interrupt	1	1
EN FLAGS	Enable Master Interrupts	1	1
SEL RB0	Select register bank 0	1	1
SEL RB1	Select register bank 1	1	1
NOP	No Operation	1	1
Registers			
INC Rr	Increment register	1	1
INC @Rr	Increment data memory	1	1
DEC Rr	Decrement register	1	1
Subroutine			
CALL addr	Jump to subroutine	2	2
RET	Return	1	2
RETR	Return and restore status	1	2
Flags			
CLR C	Clear Carry	1	1
CPL C	Complement Carry	1	1
CLR F0	Clear Flag 0	1	1
CPL F0	Complement Flag 0	1	1
CLR F1	Clear F1 Flag	1	1
CPL F1	Complement F1 Flag	1	1
Branch			
JMP addr	Jump unconditional	2	2
JMPP @A	Jump indirect	1	2
DJNZ Rr,addr	Decrement register and jump	2	2
JC addr	Jump on Carry=1	2	2
JNC addr	Jump on Carry=0	2	2
JZ addr	Jump on A Zero	2	2
JNZ addr	Jump on A not Zero	2	2
JT0 addr	Jump on T0=1	2	2
JNT0 addr	Jump on T0=0	2	2
JT1 addr	Jump on T1=1	2	2
JNT1 addr	Jump on T1=0	2	2
JF0 addr	Jump on F0 Flag=1	2	2
JF1 addr	Jump on F1 Flag=1	2	2
JTF addr	Jump on Timer Flag=1, Clear Flag	2	2
JN1BF addr	Jump on IBF Flag=0	2	2
JOBF addr	Jump on OBF Flag=1	2	2
JBb addr	Jump on Accumulator Bit	2	2

UPI™-42: 8042/8742AH UNIVERSAL PERIPHERAL INTERFACE 8-BIT SLAVE MICROCONTROLLER

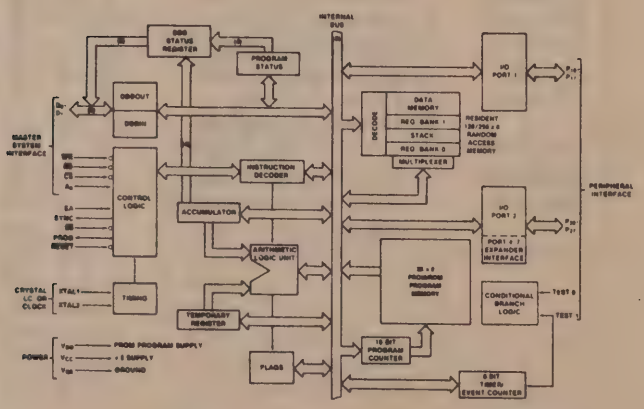
- **UPI-42: 12 MHz**
- **Pin, Software and Architecturally Compatible with 8041A/8741A**
- **8-Bit CPU plus ROM, RAM, I/O, Timer/Counter and Clock in a Single Package**
- **2048 x 8 ROM/EPROM, 128 x 8 RAM, 8-Bit Timer/Counter, 18 Programmable I/O Pins**
- **One 8-Bit Status and Two Data Registers for Asynchronous Slave-to-Master Interface**
- **DMA, Interrupt, or Polled Operation Supported**
- **Fully Compatible with all Intel and Most Other Microprocessor Families**
- **Interchangeable ROM and EPROM Versions**
- **Expandable I/O**
- **Sync Mode Available**
- **Over 90 Instructions: 70% Single Byte**
- **Available in EXPRESS — Standard Temperature Range**
- **Intelligent Programming™ Algorithm — Fastest EPROM Programming**
- **8742AH Available in 40-Lead Cerdip Package**
  8042 Available in both 40-Lead Plastic and 44-Lead Plastic Leaded Chip Carrier Packages
  (See Packaging Spec., Order #231369)

The Intel UPI-42 is a general-purpose Universal Peripheral Interface that allows the designer to develop customized solution for peripheral device control.

It is essentially a "slave" microcontroller, or a microcontroller with a slave interface included on the chip. Interface registers are included to enable the UPI device to function as a slave peripheral controller in the MCS™ Modules and iAPX family, as well as other 8-, 16-bit systems.

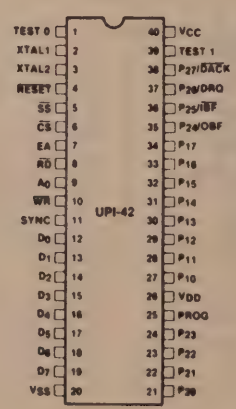
To allow full user flexibility, the program memory is available in either ROM or UV-erasable EPROM. All UPI-42 devices are fully pin compatible for easy transition from prototype to production level designs. These are the memory configurations available.

UPI Device	ROM	EPROM	RAM	Programming Voltage
8042	2K	—	256	—
8742AH	—	2K	256	12.5V



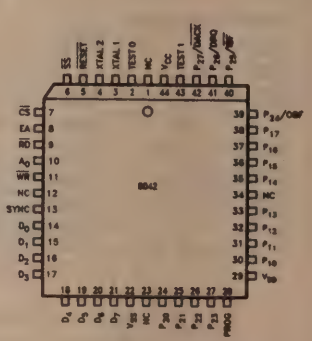
210393-1

Figure 1. Block Diagram



210393-2

Figure 2. DIP Pin Configuration



210393-3

Figure 3. PLCC Pin Configuration

Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied. Information contained herein supersedes previously published specifications on these devices from Intel.
© Intel Corporation, 1985

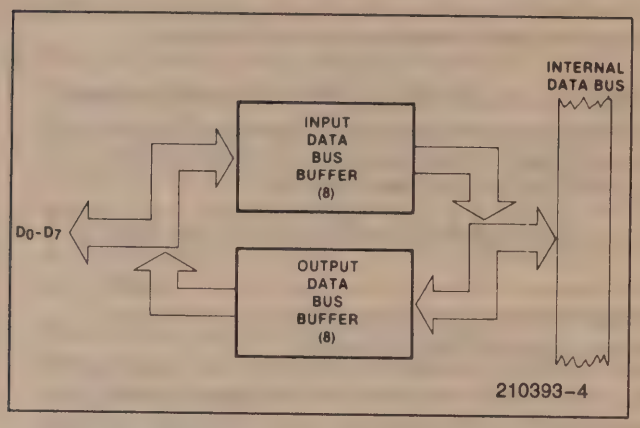
November 1985
Order Number: 210393-002

Table 1. Pin Description

Symbol	DIP Pin No.	PLCC Pin No.	Type	Name and Function
TEST 0, TEST 1	1 39	2 43	I	**TEST INPUTS:** Input pins which can be directly tested using conditional branch instructions. **FREQUENCY REFERENCE:** TEST 1 (T_1) also functions as the event timer input (under software control). TEST 0 (T_0) is used during PROM programming and verification in the 8742AH. It is also used during "sync mode" to reset the instruction state to S1 and synchronize the internal clock to PH1. See the Sync Mode Section.
XTAL 1, XTAL 2	2 3	3 4	I	**INPUTS:** Inputs for a crystal, LC or an external timing signal to determine the internal oscillator frequency.
$\overline{RESET}$	4	5	I	**RESET:** Input used to reset status flip-flops and to set the program counter to zero. $\overline{RESET}$ is also used during PROM programming and verification.
$\overline{SS}$	5	6	I	**SINGLE STEP:** Single step input used in conjunction with the SYNC output to step the program through each instruction (8742AH). This should be tied to +5V when not used. This pin is also used to put the device in synch mode by applying 12.5V to it.
$\overline{CS}$	6	7	I	**CHIP SELECT:** Chip select input used to select one UPI microcomputer out of several connected to a common data bus.
EA	7	8	I	**EXTERNAL ACCESS:** External access input which allows emulation, testing and PROM/ROM verification. This pin should be tied low if unused.
$\overline{RD}$	8	9	I	**READ:** I/O read input which enables the master CPU to read data and status words from the OUTPUT DATA BUS BUFFER or status register.
A_0	9	10	I	**COMMAND/DATA SELECT:** Address Input used by the master processor to indicate whether byte transfer is data ($A_0 = 0$, F1 is reset) or command ($A_0 = 1$, F1 is set).
$\overline{WR}$	10	11	I	**WRITE:** I/O write input which enables the master CPU to write data and command words to the UPI INPUT DATA BUS BUFFER.
SYNC	11	13	O	**OUTPUT CLOCK:** Output signal which occurs once per UPI-42 instruction cycle. SYNC can be used as a strobe for external circuitry; it is also used to synchronize single step operation.
D_0–D_7 (BUS)	12–19	14–21	I/O	**DATA BUS:** Three-state, bidirectional DATA BUS BUFFER lines used to interface the UPI-42 microcomputer to an 8-bit master system data bus.
P_{10}–P_{17}	27–34	30–33 35–38	I/O	**PORT 1:** 8-bit, PORT 1 quasi-bidirectional I/O lines. P_{10}–P_{14} and P_{17} access the signature row and security bit on the 8742AH.
P_{20}–P_{27}	21–24 35–38	24–27 39–42	I/O	**PORT 2:** 8-bit, PORT 2 quasi-bidirectional I/O lines. The lower 4 bits (P_{20}–P_{23}) interface directly to the 8243 I/O expander device and contain address and data information during PORT 4–7 access. The upper 4 bits (P_{24}–P_{27}) can be programmed to provide interrupt Request and DMA Handshake capability. Software control can configure P_{24} as Output Buffer Full (OBF) interrupt, P_{25} as Input Buffer Full ($\overline{IBF}$) interrupt, P_{26} as DMA Request (DRQ), and P_{27} as DMA ACKnowledge ($\overline{DACK}$).
PROG	25	28	I/O	**PROGRAM:** Multifunction pin used as the program pulse input during PROM programming. During I/O expander access the PROG pin acts as an address/data strobe to the 8243. This pin should be tied high if unused.
V_{CC}	40	44		**POWER:** +5V main power supply pin.
V_{DD}	26	29		**POWER:** +5V during normal operation. +12.5V during programming operation. Low power standby pin in EPROM and ROM versions.
V_{SS}	20	22		**GROUND:** Circuit ground potential.

UPI-42 FEATURES

1. Two Data Bus Buffers, one for input and one for output. This allows a much cleaner Master/Slave protocol.



2. 8 Bits of Status

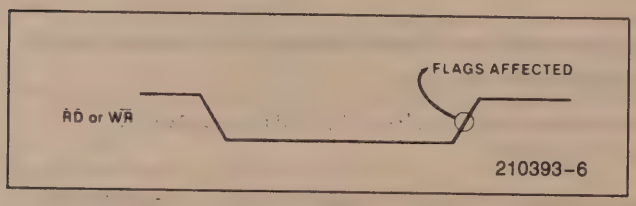
ST_7	ST_6	ST_5	ST_4	F_1	F_0	IBF	OBF
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

ST_4–ST_7 are user definable status bits. These bits are defined by the "MOV STS, A" single byte, single cycle instruction. Bits 4–7 of the acccumulator are moved to bits 4–7 of the status register. Bits 0–3 of the status register are not affected.

MOV STS, A Op Code: 90H

1	0	0	1	0	0	0	0
D_7							D_0

3. $\overline{RD}$ and $\overline{WR}$ are edge triggered. IBF, OBF, F_1 and INT change internally after the trailing edge of $\overline{RD}$ or $\overline{WR}$.

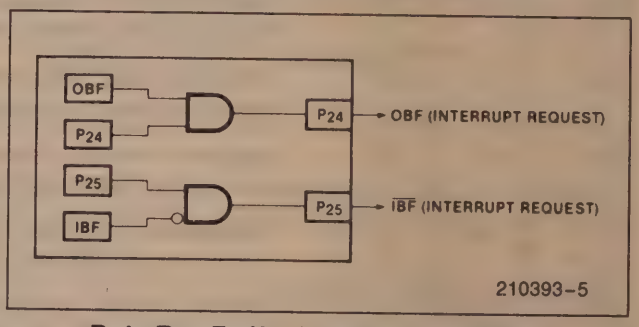


During the time that the host CPU is reading the status register, the UPI-42 is prevented from updating this register or is 'locked out.'

4. P_{24} and P_{25} are port pins or Buffer Flag pins which can be used to interrupt a master processor. These pins default to port pins on Reset.

If the "EN FLAGS" instruction has been executed, P_{24} becomes the OBF (Output Buffer Full) pin. A "1" written to P_{24} enables the OBF pin (the pin outputs the OBF Status Bit). A "0" written to P_{24} disables the OBF pin (the pin remains low). This pin can be used to indicate that valid data is available from the UPI-41A (in Output Data Bus Buffer).

If "EN FLAGS" has been executed, P_{25} becomes the $\overline{IBF}$ (Input Buffer Full) pin. A "1" written to P_{25} enables the $\overline{IBF}$ pin (the pin outputs the inverse of the IBF Status Bit. A "0" written to P_{25} disables the $\overline{IBF}$ pin (the pin remains low). This pin can be used to indicate that the UPI-42 is ready for data.



Data Bus Buffer Interrupt Capability

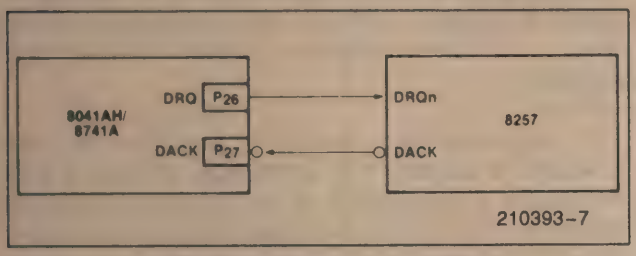
EN FLAGS Op Code: 0F5H

1	1	1	1	0	1	0	1
D_7							D_0

5. P_{26} and P_{27} are port pins or DMA handshake pins for use with a DMA controller. These pins default to port pins on Reset.

   If the "EN DMA" instruction has been executed, P_{26} becomes the DRQ (DMA Request) pin. A "1" written to P_{26} causes a DMA request (DRQ is activated). DRQ is deactivated by DACK•RD, DACK•WR, or execution of the "EN DMA" instruction.

   If "EN DMA" has been executed, P_{27} becomes the $\overline{DACK}$ (DMA ACKnowledge) pin. This pin acts as a chip select input for the Data Bus Buffer registers during DMA transfers.



DMA Handshake Capability

EN DMA Op Code: 0E5H

1	1	1	0	0	1	0	1

D_7 D_0

6. When EA is enabled on the UPI-42, the program counter is placed on Port 1 and the lower three bits of Port 2 (MSB = P_{22}, LSB = P_{10}). On the UPI-42 this information is multiplexed with PORT DATA (see port timing diagrams at end of this data sheet).
7. The 8742AH supports the intelligent Programming Algorithm. (See the Programming Section.)

APPLICATIONS

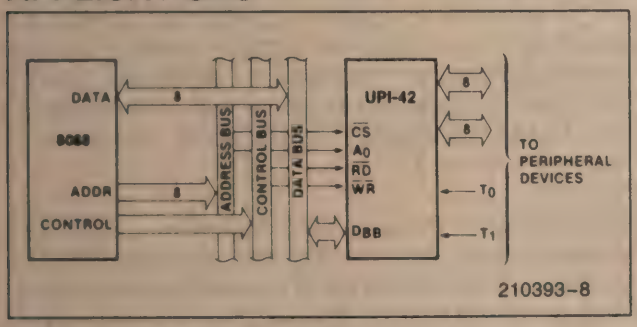


Figure 3. 8088-UPI-42 Interface

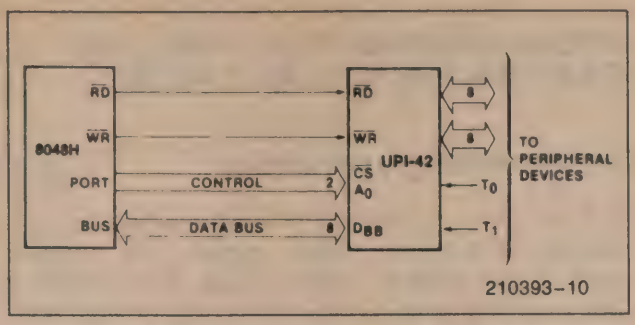


Figure 4. 8048H-UPI-42 Interface

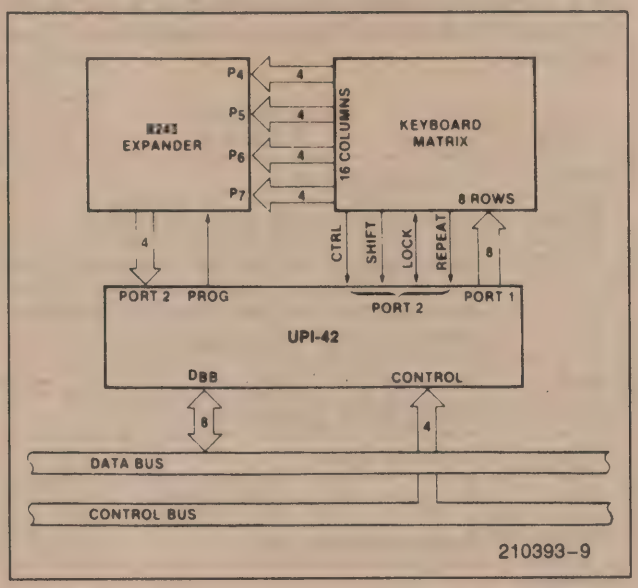


Figure 5. UPI-42-8243 Keyboard Scanner

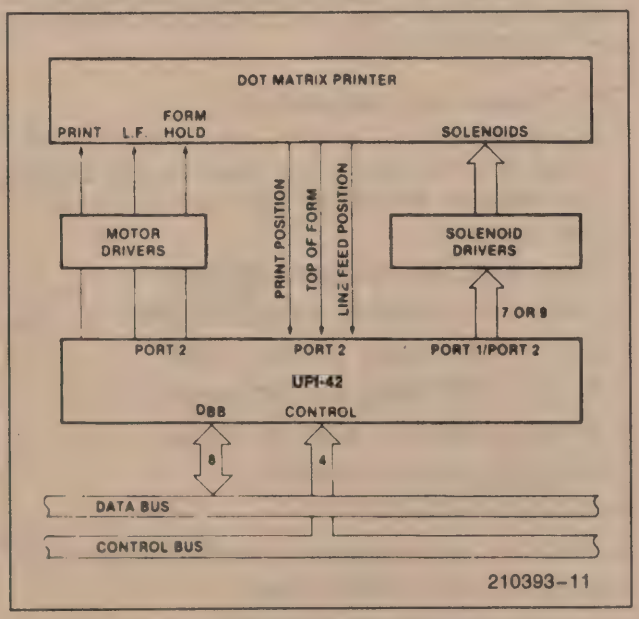


Figure 6. UPI-42 80-Column Matrix Printer Interface

PROGRAMMING, VERIFYING, AND ERASING THE 8742AH EPROM

Programming Verification

In brief, the programming process consists of: activating the program mode, applying an address, latching the address, applying data, and applying a programming pulse. Each word is programmed completely before moving on to the next and is followed by a verification step. The following is a list of the pins used for programming and a description of their functions:

Pin	Function
XTAL 1	2 Clock Inputs
$\overline{\text{Reset}}$	Initialization and Address Latching
Test 0	Selection of Program or Verify Mode
EA	Activation of Program/Verify Signature Row/Security Bit Modes
BUS	Address and Data Input Data Output During Verify
P_{20-22}	Address Input
V_{DD}	Programming Power Supply
PROG	Program Pulse Input

WARNING
An attempt to program a missocketed 8742AH will result in severe damage to the part. An indication of a properly socketed part is the appearance of the SYNC clock output. The lack of this clock may be used to disable the programmer.

The Program/Verify sequence is:

1. A_0 = 0V, CS = 5V, EA = 5V, RESET = 0V, TEST O = 5V, V_{DD} = 5V, clock applied or internal oscillator operating, BUS floating, PROG = 5V.
2. Insert 8742AH in programming socket
3. TEST 0 = 0V (select program model)
4. EA = 12.5V (active program mode)
5. V_{DD} = 12.5V (programming power)
6. Address applied to BUS and P_{20-22}
7. $\overline{\text{RESET}}$ = 5V (latch address)
8. Data applied to BUS
9. PROG = V_{CC} followed by one 1 ms pulse to 0V
10. TEST 0 = 5V (verify mode)
11. Read and verify data on BUS
12. TEST 0 = 0V
13. $\overline{\text{RESET}}$ = 0V and repeat from step 6
14. Programmer should be at conditions of step 1 when 8742AH is removed from socket

Please follow the int$_e$ligent Programming flow chart for proper programming procedure.

8742AH Erasure Characteristics

The erasure characteristics of the 8742AH are such that erasure begins to occur when exposed to light with wavelengths shorter than approximately 4000 Angstroms (Å). It should be noted that sunlight and certain types of fluorescent lamps have wavelengths in the 3000-4000Å range. Data show that constant exposure to room level fluorescent lighting could erase the typical 8742AH in approximately 3 years while it would take approximately one week to cause erasure when exposed to direct sunlight. If the 8742AH is to be exposed to these types of lighting conditions for extended periods of time, opaque labels are available from Intel which should be placed over the 8742AH window to prevent unintentional erasure.

The recommended erasure procedure for the 8742AH is exposure to shortwave ultraviolet light which has a wavelength of 2537Å. The integrated dose (i.e., UV intensity × exposure time) for erasure should be a minimum of 15 w-sec/cm^2. The erasure time with this dosage is approximately 15 to 20 minutes using an ultraviolet lamp with a 12,000 μW/cm^2 power rating. The 8742AH should be placed within one inch of the lamp tubes during erasure. Some lamps have a filter on their tubes which should be removed before erasure. Exposure of the 8742AH to high intensity UV light for long periods may cause permanent damage.

int$_e$ligent Programming™ Algorithm

The 8742AH int$_e$ligent Programming Algorithm rapidly programs Intel 8742AH EPROMs using an efficient and reliable method particularly suited to the production programming environment. Typical programming time for individual devices is on the order of 10 seconds. Programming reliability is also ensured as the incremental program margin of each byte is continually monitored to determine when it has been successfully programmed. A flowchart of the 8742AH int$_e$ligent Programming Algorithm is shown in Figure 7.

The int$_e$ligent Programming Algorithm utilizes two different pulse types: initial and overprogram. The duration of the initial PROG pulse(s) is one millisecond, which will then be followed by a longer overprogram pulse of length 3X msec. X is an iteration counter and is equal to the number of the initial one millisecond pulses applied to a particular 8742AH location, before a correct verify occurs. Up to 25 one-millisecond pulses per byte are provided for before the overprogram pulse is applied.

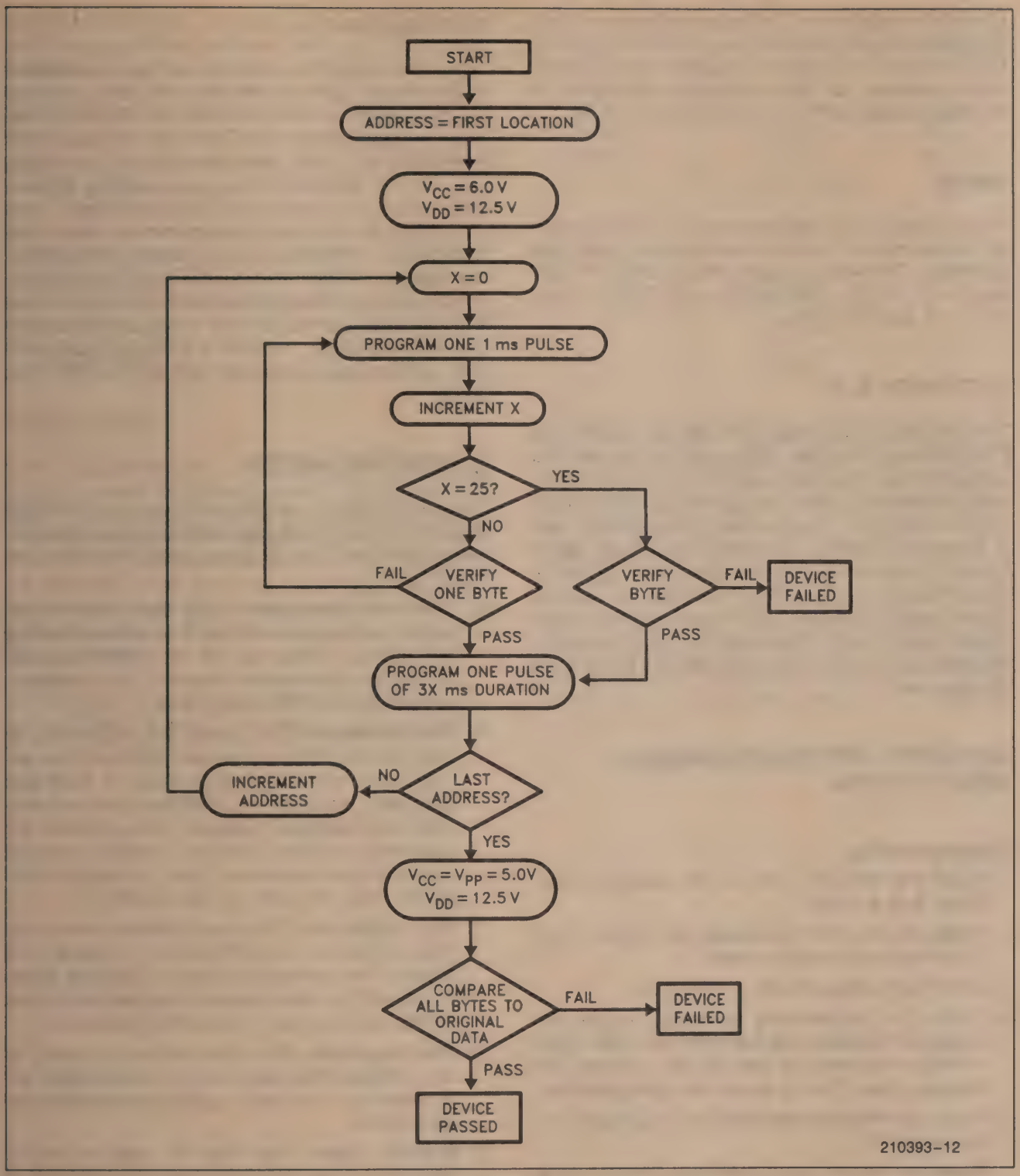

START
ADDRESS = FIRST LOCATION
V_{CC} = 6.0 V
V_{DD} = 12.5 V
X = 0
PROGRAM ONE 1 ms PULSE
INCREMENT X
X = 25?
YES
NO
VERIFY ONE BYTE
FAIL
PASS
VERIFY BYTE
FAIL
PASS
DEVICE FAILED
PROGRAM ONE PULSE OF 3X ms DURATION
LAST ADDRESS?
NO
INCREMENT ADDRESS
YES
V_{CC} = V_{PP} = 5.0V
V_{DD} = 12.5 V
COMPARE ALL BYTES TO ORIGINAL DATA
FAIL
DEVICE FAILED
PASS
DEVICE PASSED
210393-12

Figure 7

The entire sequence of program pulses and byte verifications is performed at V_{CC} = 6.0V and V_{DD} = 12.5V. When the intelligent Programming cycle has been completed, all bytes should be compared to the original data with V_{CC} = 5.0, V_{DD} = 12.5V.

Verify

A verify should be performed on the programmed bits to determine that they have been correctly programmed. The verify is performed with T0 = 5V, V_{DD} = 12.5V, EA = 12.5V, $\overline{SS}$ = 5V, $\overline{PROG}$ = 5V, A0 = 0V, and $\overline{CS}$ = 5V.

SECURITY BIT

The security bit is a single EPROM cell outside the EPROM array. The user can program this bit with the appropriate access code and the normal programming procedure, to inhibit any external access to the EPROM contents. Thus the user's resident program is protected. There is no direct external access to this bit. However, the security byte in the signature mode has the same address and can be used to check indirectly whether the security bit has been programmed or not. The security bit has no effect on the signature mode, so the security byte can always be examined.

SECURITY BIT PROGRAMMING/ VERIFICATION

Programming :

a. Read the security byte of the signature row. Make sure it is 00H.

b. Apply access code to appropriate inputs to put the device into security mode.

c. Apply high voltage to EA and V_{DD} pins.

d. Follow the programming procedure as per the intelligent Programming Algorithm with DB0–DB7 = high. Not only the security bit, but also the security byte of the signature row is programmed.

e. Verify that the security byte of the signature row contains FFH.

f. Read two known bytes from the EPROM array and verify that the wrong data are retrieved in at least one verification. If the EPROM can still be read, the security bit may have not been fully programmed though the security byte in the signature row has.

Verification:

Since the security bit address overlaps the address of the security byte of the signature row, it can be used to check indirectly whether the security bit has been programmed or not. Therefore, the security bit verification is a mere read operation of the security byte of the signature row (1 = security bit programmed; 0 = security bit unprogrammed). Note that during the security bit programming, reading security byte = FFH does not necessarily indicate that the security bit has been successfully programmed. Thus, it is recommended that two known bytes in the EPROM array be read and the wrong data should be read at least once, because it is highly improbable that random data coincides with the correct ones twice.

SIGNATURE MODE

The UPI-42 has an additional 32 bytes of EPROM available for Intel and user signatures and miscellaneous purposes. The 32 bytes are partitioned as follows:

A. **Test code/checksum**—This ROM memory can accommodate up to 25 bytes of code for testing the internal nodes that are not testable by executing from the external memory. The checksum is used in the ROM testing only.

B. **Intel signature**—This allows the programmer to read from the UPI-42 the manufacturer of the device and the exact product name. It facilitated automatic selection of EPROM size and programming voltages. Location 10H contains the manufacturer code. For Intel, it is 89H. Location 11H contains the device code. The code is 43H for the 8042; it is 42H for the 8742AH.

C. **User signature**—The user signature memory is implemented in the EPROM and consists of 2 bytes for the customer to program his own signature code (for identification purposes and quick sorting of previously programmed materials).

D. **Test signature (B)**—This memory is used to store testing information such as: test data, bin number, etc. (for use in quality and manufacturing control).

E. **Security byte**—This byte is used to check whether the security bit has been programmed or not (see the security bit section).

The signature mode can be accessed by setting P10=0, P11=1, P12=1, P13=1, P14=1, and then following the programming and/or verification procedures. The location of the various address partitions are as follows:

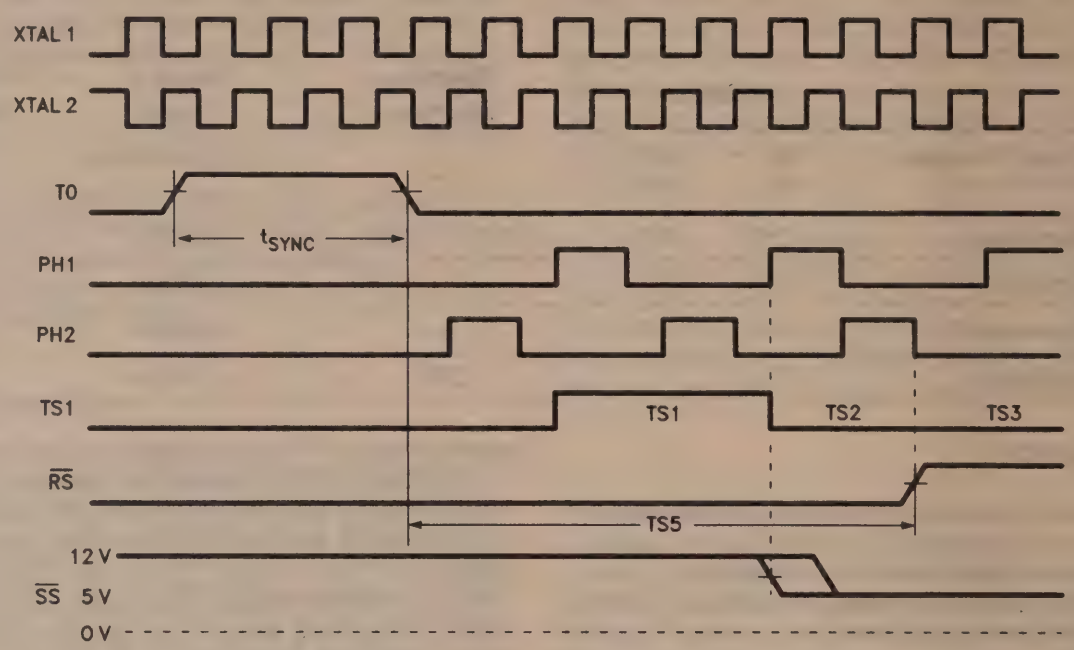
	Address		Alternate Address		Device Type	No. of Bytes
Test Code/Checksum	0 16H	0FH 1EH	—		EPROM/ROM	25
Intel Signature	10H	11H	D0H	D1H	ROM	2
User Signature	12H	13H	—		EPROM	2
Test Signature (B)	14H	15H	—		EPROM	2
Security Byte	1FH		DFH		EPROM	1

SYNC MODE

SYNC mode allows the UPI-42 to be forced externally into a known state upon reset. It can be used to synchronize multiple parts on a printed circuit board. It clears the oscillator prescaler, the phases, and the Time State Generator. Since, under normal operation, the RS signal can occur asynchronously with respect to the UPI-42's internal state, it cannot be guaranteed that a $\overline{RS}$ will leave several devices in the same state.

SYNC mode is activated when the $\overline{SS}$ pin is at a voltage level of 12V. The actual synchronization starts when the T0 pin is raised to VIH during XTAL 1=0. After 3 XTAL 1 pulses, the prescaler is forced into a known state, which sets both PH1 and PH2 to a "0" state as shown. The Time State Generator is completely reset with TS1=TS2=TS3=TS4=TS5=0, and the input to TS1 is equal to a "1" state. SYNC is removed when T0 is brought from "1" to "0" during the next XTAL=0. The subsequent PH2=1 and PH1=0, then PH2=0, and PH1=1 will latch a "1" into TS1. A normal reset is then given to put the device into proper operation. What state the device will be in after the RS becomes a "1" can be determined by tracking how many XTAL 1 cycles occur during RS.

SYNC MODE TIMING DIAGRAMS



210393-13

Minimum Specifications
SYNC Operation Time, t_{SYNC} = 3.5 XTAL 1 Clock cycles. Reset Time, t_{RS} = 4 t_{CY}.

NOTE:
The rising and falling edges of T0 should not coincide with the falling edge of XTAL1 clock.

ACCESS CODE

The following table summarizes the access codes required to invoke the SYNC mode, signature mode, and the security bit, respectively. Also, the programming and verification modes are included for comparison.

Modes		Control Signals							Data Bus								Access Code									
																	Port 2			Port 1						
		T0	RST	SS	EA	PROG	V_{DD}	V_{CC}	0	1	2	3	4	5	6	7	0	1	2	0	1	2	3	4	7
Programming Mode		0	0	1	HV	1	V_{DOH}	V_{CC}	Address								Address			00/11	X	X	X	X	X
		0	1	1	HV	STB	V_{DOH}	V_{CC}	Data In								Address								
Verification Mode		1	0	1	HV	1	V_{DOH}	V_{CC}	Address								Address			00/11	X	X	X	X	X
		1	1	1	HV	1	V_{DOH}	V_{CC}	Data Out								Address								
Sync Mode		STB High	0	HV	0	X	V_{CC}	V_{CC}	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Signature Mode	Prog	0	0	1	HV	1	V_{DOH}	V_{CC}	Addr. (See Sig Row Table)								0	0	0	0	1	1	1	1	1
		0	1	1	HV	STB	V_{DOH}	V_{CC}	Data In								0	0	0						
	Verify	1	0	1	HV	1	V_{DOH}	V_{CC}	Addr. (See Sig Row Table)								0	0	0						
		1	1	1	HV	1	V_{DOH}	V_{CC}	Data Out								0	0	0						
Security Bit	Prog	0	0	1	HV	1	V_{DOH}	V_{CC}	1	1	1	1	1	1	1	1	0	0	0						
		0	1	1	HV	STB	V_{DOH}	V_{CC}	1	1	1	1	1	1	1	1	0	0	0						

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias ....0°C to +70°C

Storage Temperature .....−65°C to +150°C

Voltage on Any Pin with Respect to Ground.....−0.5V to +7V

Power Dissipation .....1.5 W

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS $T_A = 0°C$ to $+70°C$, $V_{CC} = V_{DD} = +5V \pm 10\%$

Symbol	Parameter	8042/8742		Units	Notes
		Min	Max		
V_{IL}	Input Low Voltage (Except XTAL1, XTAL2, RESET)	−0.5	0.8	V	
V_{IL1}	Input Low Voltage (XTAL1, XTAL2, RESET)	−0.5	0.6	V	
V_{IH}	Input High Voltage (Except XTAL1, XTAL2, RESET)	2.0	V_{CC}	V	
V_{IH1}	Input High Voltage (XTAL1, RESET)	3.5	V_{CC}	V	
V_{IH2}	Input High Voltage (XTAL2)	2.2	V_{CC}	V	
V_{OL}	Output Low Voltage (D_0–D_7)		0.45	V	$I_{OL} = 2.0$ mA
V_{OL1}	Output Low Voltage ($P_{10}P_{17}$, $P_{20}P_{27}$, Sync)		0.45	V	$I_{OL} = 1.6$ mA
V_{OL2}	Output Low Voltage (PROG)		0.45	V	$I_{OL} = 1.0$ mA
V_{OH}	Output High Voltage (D_0–D_7)	2.4		V	$I_{OH} = -400\ \mu A$
V_{OH1}	Output High Voltage (All Other Outputs)	2.4			$I_{OH} = -50\ \mu A$
I_{IL}	Input Leakage Current (T_0, T_1, RD, WR, CS, A_0, EA)		±10	μA	$V_{SS} \le V_{IN} \le V_{CC}$
I_{OFL}	Output Leakage Current (D_0–D_7, High Z State)		±10	μA	$V_{SS} + 0.45 \le V_{OUT} \le V_{CC}$
I_{LI}	Low Input Load Current ($P_{10}P_{17}$, $P_{20}P_{27}$)		0.3	mA	$V_{IL} = 0.8V$
I_{LI1}	Low Input Load Current (RESET, SS)		0.2	mA	$V_{IL} = 0.8V$
I_{DD}	V_{DD} Supply Current		20	mA	Typical = 5 mA
$I_{CC} + I_{DD}$	Total Supply Current		135	mA	Typical = 60 mA
I_{IH}	Input Leakage Current (P_{10}–P_{17}, P_{20}–P_{27})		100	μA	$V_{IN} = V_{CC}$
C_{IN}	Input Capacitance		10	pF	
C_{IO}	I/O Capacitance		20	pF	

D.C. CHARACTERISTICS—PROGRAMMING

$T_A = 25°C \pm 5°C$, $V_{CC} = 6V \pm 0.25V$, $V_{DD} = 12.5V \pm 0.5V$

Symbol	Parameter	Min	Max	Units
V_{DOH}	V_{DD} Program Voltage High Level	12	13	V(1)
V_{DDL}	V_{DD} Voltage Low Level	4.75	5.25	V
V_{PH}	PROG Program Voltage High Level	2.0	5.5	V
V_{PL}	PROG Voltage Low Level	−0.5	0.8	V
V_{EAH}	Input High Voltage for EA, SS	12.0	13.0	V(2)
V_{EAL}	EA Voltage Low Level		5.25	V
I_{DD}	V_{DD} High Voltage Supply Current		30.0	mA
I_{PROG}	PROG HIgh Voltage Supply Current		1.0	mA
I_{EA}	EA Voltage Supply Current		1.0	mA

NOTES:
1. Voltages over 13V applied to pin V_{DD} will permanently damage the device.
2. V_{EAH} must be applied to EA before V_{DOH} and removed after V_{DDL}.
3. V_{CC} must be applied simultaneously or before V_{DD} and must be removed simultaneously or after V_{DD}.

A.C. CHARACTERISTICS $T_A = 0°C$ to $+70°C$, $V_{SS} = 0V$, $V_{CC} = V_{DD} = +5V \pm 10\%$

DBB READ

Symbol	Parameter	Min	Max	Units
t_{AR}	CS, A_0 Setup to RD ↓	0		ns
t_{RA}	CS, A_0 Hold After RD ↑	0		ns
t_{RR}	RD Pulse Width	160		ns
t_{AD}	CS, A_0 to Data Out Delay		130	ns
t_{RD}	RD ↓ to Data Out Delay		130	ns
t_{DF}	RD ↑ to Data Float Delay		85	ns

DBB WRITE

Symbol	Parameter	Min	Max	Units
t_{AW}	CS, A_0 Setup to WR ↓	0		ns
t_{WA}	CS, A_0 Hold After WR ↑	0		ns
t_{WW}	WR Pulse Width	160		ns
t_{DW}	Data Setup to WR ↑	130		ns
t_{WD}	Data Hold After WR ↑	0		ns

CLOCK

Symbol	Parameter	Min	Max	Units
t_{CY}	Cycle Time	1.25	9.20	μs(1)
t_{CYC}	Clock Period	83.3	613	ns
t_{PWH}	Clock High Time	33		ns
t_{PWL}	Clock Low Time	33		ns
t_R	Clock Rise Time		10	ns
t_F	Clock Fall Time		10	ns

NOTE:
1. $t_{CY} = 15/f(XTAL)$

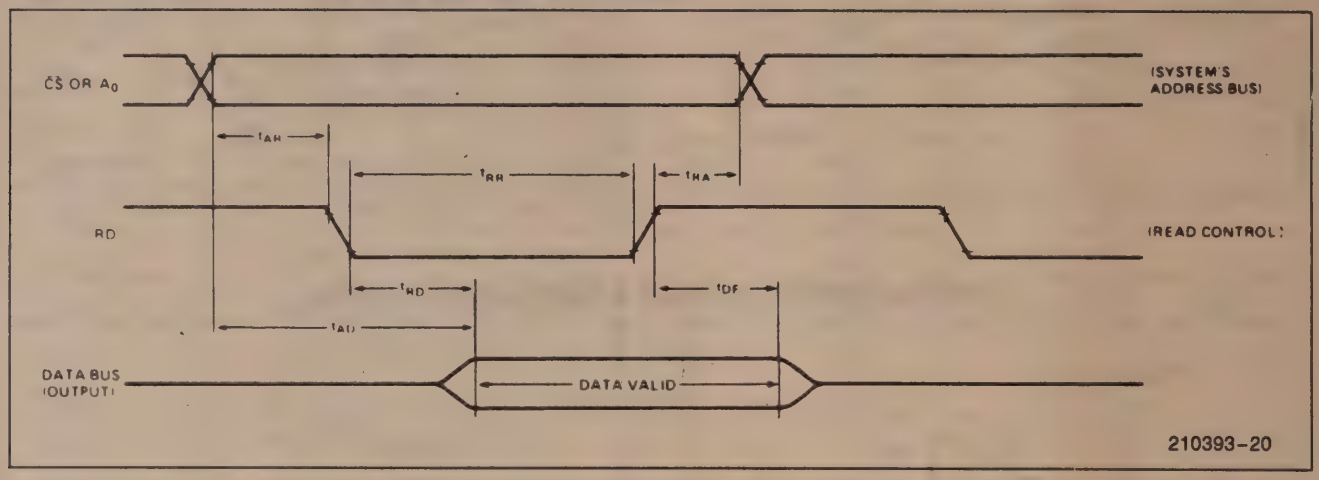
A.C. CHARACTERISTICS DMA

Symbol	Parameter	Min	Max	Units
t_{ACC}	DACK to WR or RD	0		ns
t_{CAC}	RD or WR to DACK	0		ns
t_{ACD}	DACK to Data Valid		130	ns
t_{CRQ}	RD or WR to DRQ Cleared		110	ns(1)

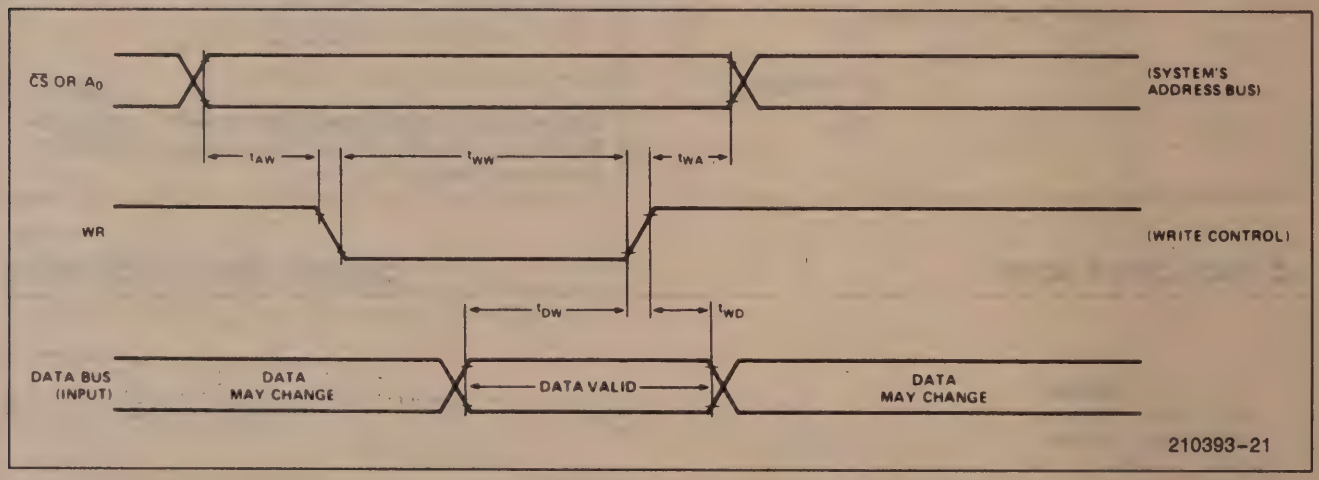
NOTE:
1. $C_L = 150$ pF.

WAVEFORMS

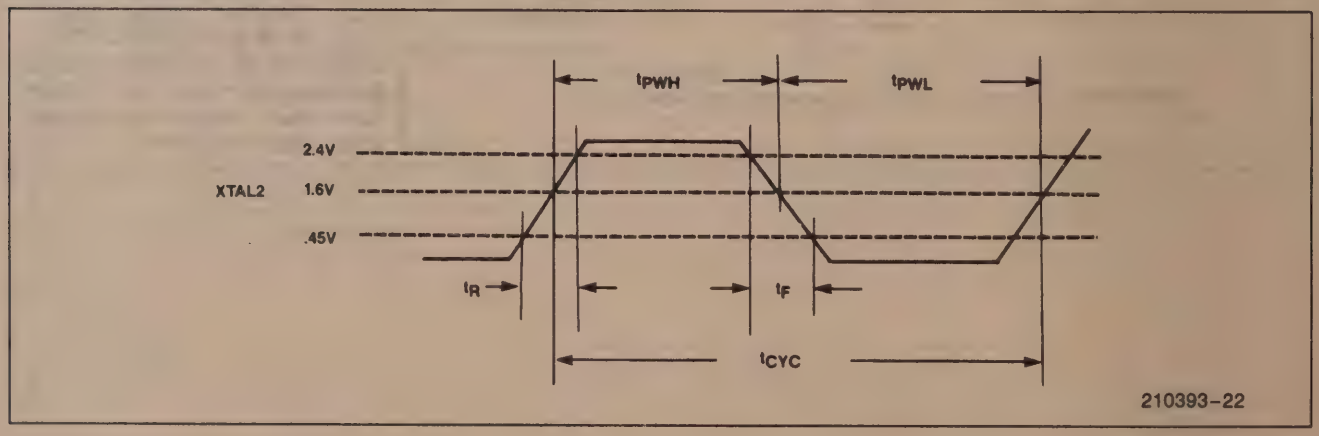
READ OPERATION—DATA BUS BUFFER REGISTER



WRITE OPERATION—DATA BUS BUFFER REGISTER

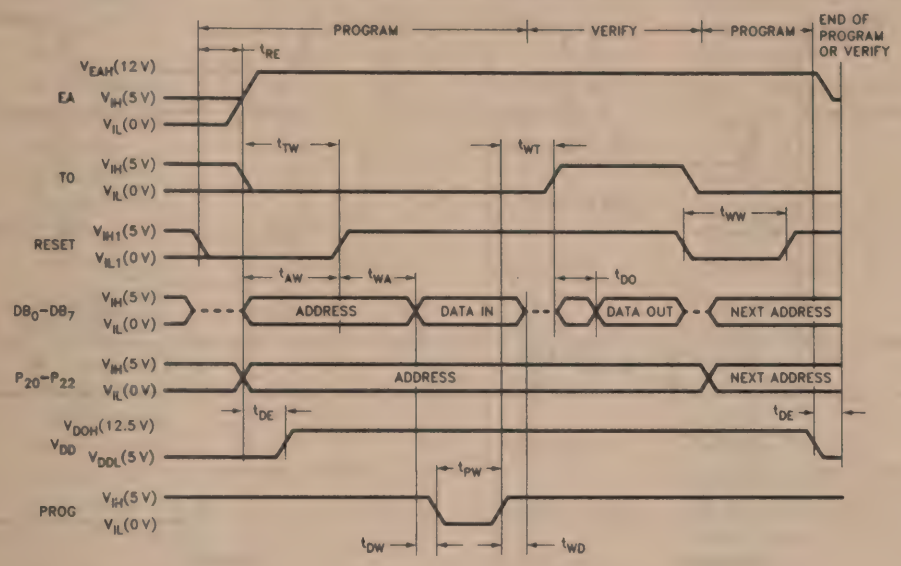


CLOCK TIMING



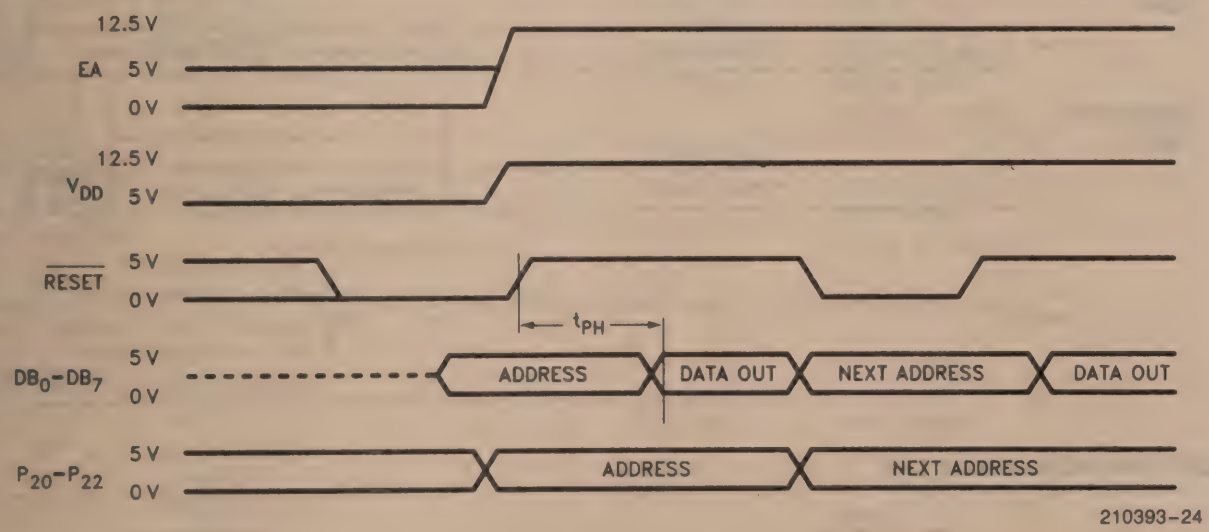
WAVEFORMS (Continued)

COMBINATION PROGRAM/VERIFY MODE



NOTES:
1. A_0 must be held low (0V) during program/verify modes.
2. For V_{IH}, V_{IH1}, V_{IL}, V_{IL1}, V_{DOH}, and V_{DDL}, please consult the D.C. Characteristics Table.
3. When programming the 8742AH, a 0.1 μF capacitor is required across V_{DD} and ground to suppress spurious voltage transients which can damage the device.

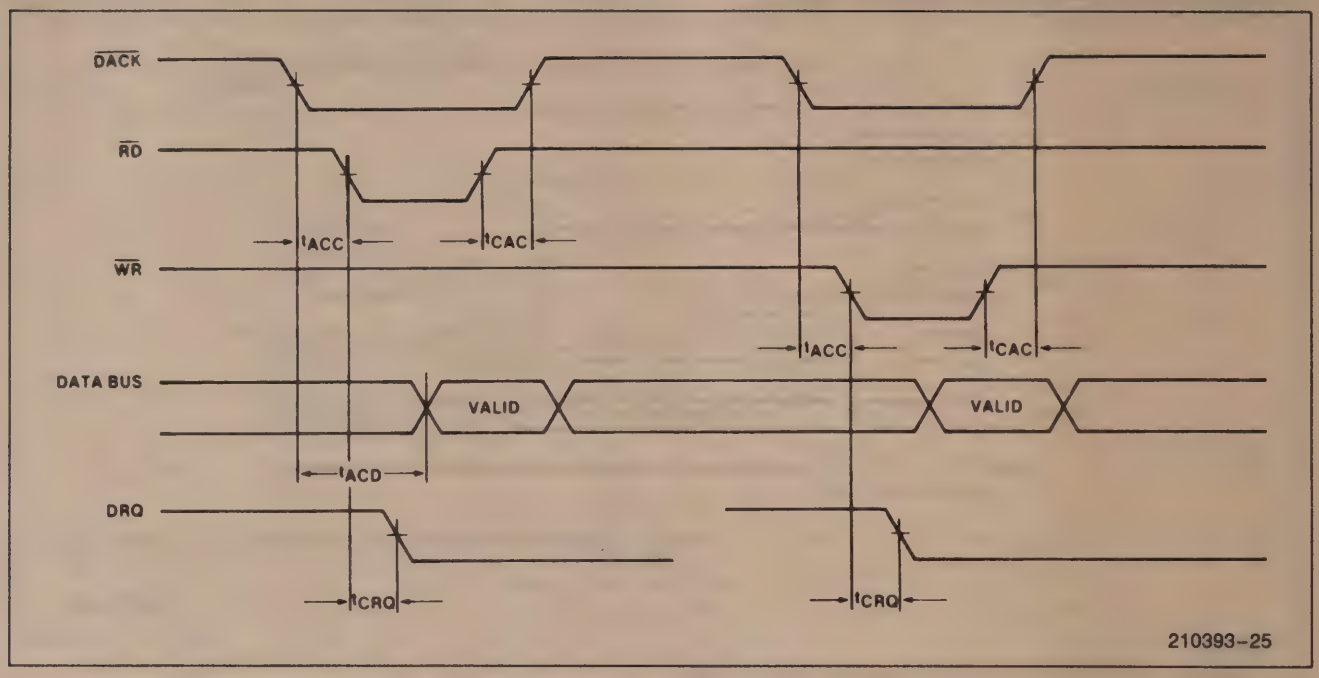
VERIFY MODE



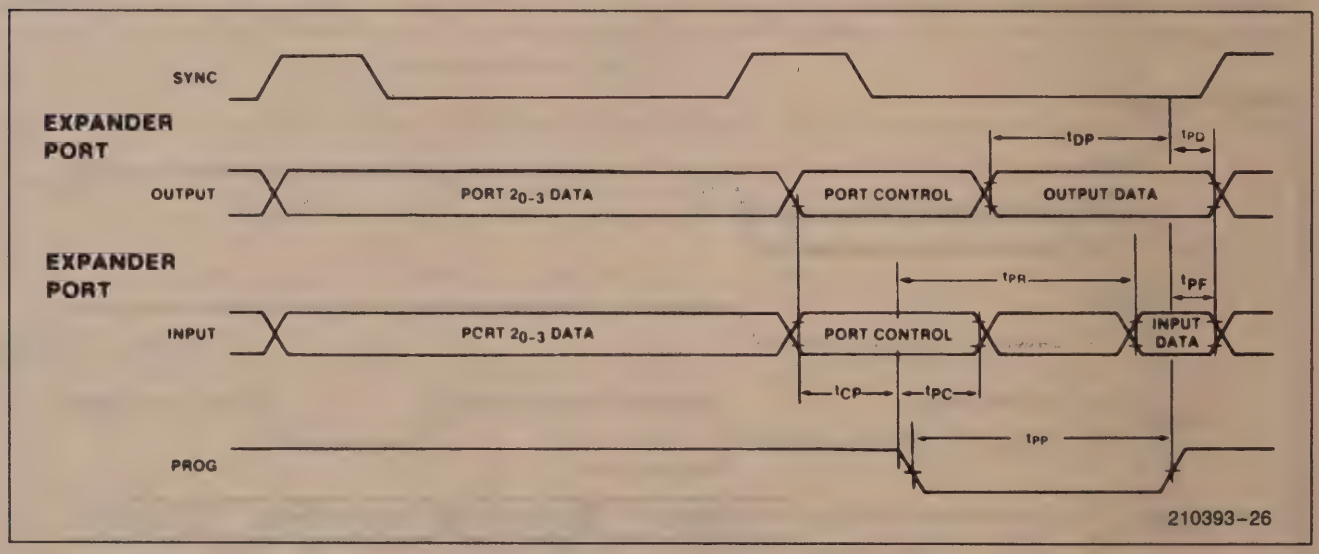
NOTES:
1. PROG must float if EA is low.
2. PROG must always float for 8042.
3. TO must be held high (5V).
4. P_{10}–P_{17} = 5V or must float.
5. P_{24}–P_{27} = 5V or must float.
6. A_0 must be held low during programming/verify modes.

WAVEFORMS (Continued)

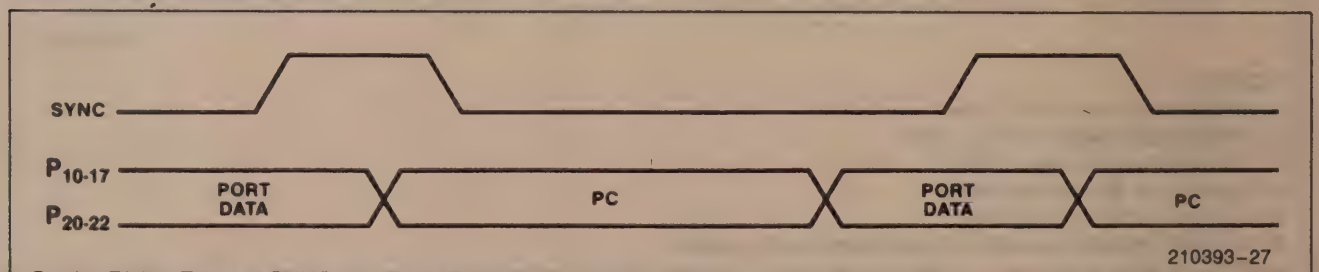
DMA



PORT 2



PORT TIMING DURING EA



On the Rising Edge of SYNC and EA is Enabled, Port Data is Valid and can be Strobed on the Trailing Edge of Sync the Program Counter Contents are Available.

Table 2. UPI™ Instruction Set

Mnemonic	Description	Bytes	Cycles
ACCUMULATOR			
ADD A, Rr	Add register to A	1	1
ADD A, @Rr	Add data memory to A	1	1
ADD A, #data	Add immediate to A	2	2
ADDC A, Rr	Add register to A with carry	1	1
ADDC A, @Rr	Add data memory to A with carry	1	1
ADDC A, #data	Add immediate to A with carry	2	2
ANL A, Rr	AND register to A	1	1
ANL, A @Rr	AND data memory to A	1	1
ANL A, #data	AND immediate to A	2	2
ORL A, Rr	OR register to A	1	1
ORL, A, @Rr	OR data memory to A	1	1
ORL A, #data	OR immediate to A	2	2
XRL A, Rr	Exclusive OR register to A	1	1
XRL A, @Rr	Exclusive OR data memory to A	1	1
XRL A, #data	Exclusive OR immediate to A	2	2
INC A	Increment A	1	1
DEC A	Decrement A	1	1
CLR A	Clear A	1	1
CPL A	Complement A	1	1
DA A	Decimal Adjust A	1	1
SWAP A	Swap nibbles of A	1	1
RL A	Rotate A left	1	1
RLC A	Rotate A left through carry	1	1
RR A	Rotate A right	1	1
RRC A	Rotate A right through carry	1	1
INPUT/OUTPUT			
IN A, Pp	Input port to A	1	2
OUTL Pp, A	Output A to port	1	2
ANL Pp, #data	AND immediate to port	2	2
ORL Pp, #data	OR immediate to port	2	2
IN A, DBB	Input DBB to A, clear IBF	1	1
OUT DBB, A	Output A to DBB, set OBF	1	1
MOV STS, A	A_4–A_7 to Bits 4–7 of Status	1	1
MOVD A, Pp	Input Expander port to A	1	2
MOVD Pp, A	Output A to Expander port	1	2
ANLD Pp, A	AND A to Expander port	1	2
ORLD Pp, A	OR A to Expander port	1	2

Mnemonic	Description	Bytes	Cycles
DATA MOVES			
MOV A, Rr	Move register to A	1	1
MOV A, @Rr	Move data memory to A	1	1
MOV A, #data	Move immediate to A	2	2
MOV Rr, A	Move A to register	1	1
MOV @Rr, A	Move A to data memory	1	1
MOV Rr, #data	Move immediate to register	2	2
MOV @Rr, #data	Move immediate to data memory	2	2
MOV A, PSW	Move PSW to A	1	1
MOV PSW, A	Move A to PSW	1	1
XCH A, Rr	Exchange A and register	1	1
XCH A, @Rr	Exchange A and data memory	1	1
XCHD A, @Rr	Exchange digit of A and register	1	1
MOVP A, @A	Move to A from current page	1	2
MOVP3, A, @A	Move to A from page 3	1	2
TIMER/COUNTER			
MOV A, T	Read Timer/Counter	1	1
MOV T, A	Load Timer/Counter	1	1
STRT T	Start Timer	1	1
STRT CNT	Start Counter	1	1
STOP TCNT	Stop Timer/Counter	1	1
EN TCNTI	Enable Timer/Counter Interrupt	1	1
DIS TCNTI	Disable Timer/Counter Interrupt	1	1
CONTROL			
EN DMA	Enable DMA Handshake Lines	1	1
EN I	Enable IBF Interrupt	1	1
DIS I	Diable IBF Interrupt	1	1
EN FLAGS	Enable Master Interrupts	1	1
SEL RB0	Select register bank 0	1	1
SEL RB1	Select register bank 1	1	1
NOP	No Operation	1	1
REGISTERS			
INC Rr	Increment register	1	1
INC @Rr	Increment data memory	1	1
DEC Rr	Decrement register	1	1

Table 2. UPI™ Instruction Set (Continued)

Mnemonic	Description	Bytes	Cycles
SUBROUTINE			
CALL addr	Jump to subroutine	2	2
RET	Return	1	2
RETR	Return and restore status	1	2
FLAGS			
CLR C	Clear Carry	1	1
CPL C	Complement Carry	1	1
CLR F0	Clear Flag 0	1	1
CPL F0	Complement Flag 0	1	1
CLR F1	Clear F1 Flag	1	1
CPL F1	Complement F1 Flag	1	1
BRANCH			
JMP addr	Jump unconditional	2	2
JMPP @A	Jump indirect	1	2
DJNZ Rr, addr	Decrement register and jump	2	2
JC addr	Jump on Carry = 1	2	2
JNC addr	Jump on Carry = 0	2	2
JZ addr	Jump on A Zero	2	2
JNZ addr	Jump on A not Zero	2	2
JT0 addr	Jump on T0 = 1	2	2
JNT0 addr	Jump on T0 = 0	2	2
JT1 addr	Jump on T1 = 1	2	2
JNT1 addr	Jump on T1 = 0	2	2
JF0 addr	Jump on F0 Flag = 1	2	2
JF1 addr	Jump on F1 Flag = 1	2	2
JTF addr	Jump on Timer Flag = 1, Clear Flag	2	2
JNIBF addr	Jump on IBF Flag = 0	2	2
JOBF addr	Jump on OBF Flag = 1	2	2
JBb addr	Jump on Accumula-for Bit	2	2

intel

8243
MCS-48® INPUT/OUTPUT EXPANDER

- Low Cost
- Simple Interface to MCS-48® Microcomputers
- Four 4-Bit I/O Ports
- AND and OR Directly to Ports
- 24-Pin DIP
- Single 5V Supply
- High Output Drive
- Direct Extension of Resident 8048 I/O Ports

The Intel® 8243 is an input/output expander designed specifically to provide a low cost means of I/O expansion for the MCS-48® family of single chip microcomputers. Fabricated in 5 volts NMOS, the 8243 combines low cost, single supply voltage and high drive current capability.

The 8243 consists of four 4-bit bidirectional static I/O ports and one 4-bit port which serves as an interface to the MCS-48 microcomputers. The 4-bit interface requires that only 4 I/O lines of the 8048 be used for I/O expansion, and also allows multiple 8243's to be added to the same bus.

The I/O ports of the 8243 serve as a direct extension of the resident I/O facilities of the MCS-48 microcomputers and are accessed by their own MOV, ANL, and ORL instructions.

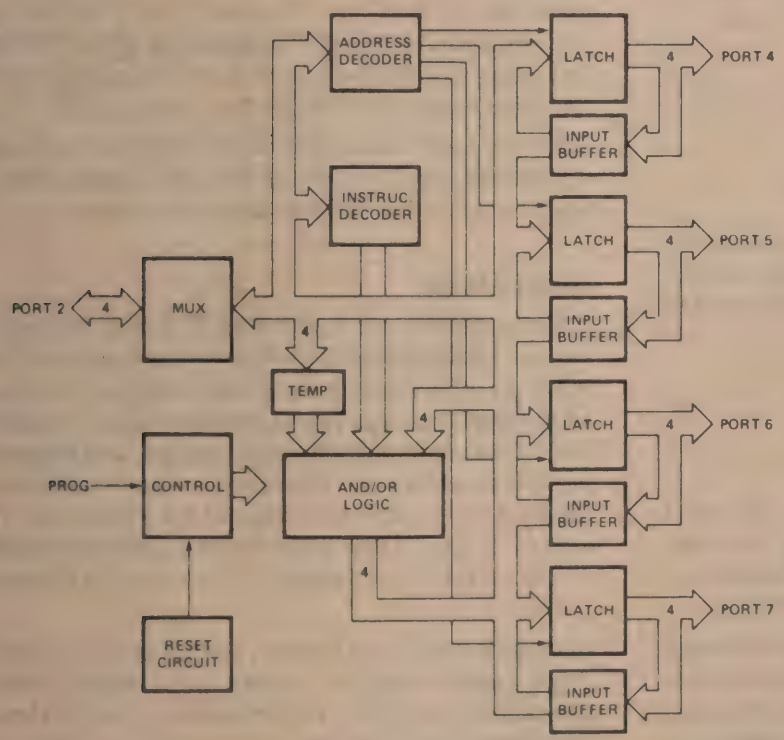


Figure 1. 8243 Block Diagram

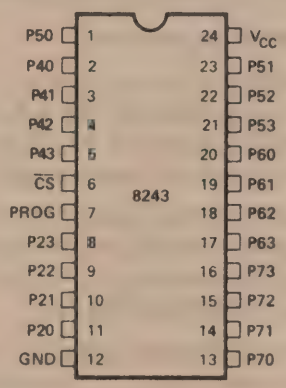


Figure 2. 8243 Pin Configuration

Intel Corporation Assumes No Responsibility for the Use of Any Circuitry Other Than Circuitry Embodied in an Intel Product. No Other Circuit Patent Licenses are Implied.
INTEL CORPORATION, 1980

Table 1. Pin Description

Symbol	Pin No.	Function
PROG	7	Clock Input. A high to low transition on PROG signifies that address and control are available on P20-P23, and a low to high transition signifies that data is available on P20-P23.
$\overline{CS}$	6	Chip Select Input. A high on CS inhibits any change of output or internal status.
P20-P23	11-8	Four (4) bit bi-directional port contains the address and control bits on a high to low transition of PROG. During a low to high transition contains the data for a selected output port if a write operation, or the data from a selected port before the low to high transition if a read operation.
GND	12	0 volt supply.
P40-P43 P50-P53 P60-P63 P70-P73	2-5 1, 23-21 20-17 13-16	Four (4) bit bi-directional I/O ports. May be programmed to be input (during read), low impedance latched output (after write), or a tri-state (after read). Data on pins P20-P23 may be directly written, ANDed or ORed with previous data.
V_{CC}	24	+5 volt supply.

FUNCTIONAL DESCRIPTION

General Operation

The 8243 contains four 4-bit I/O ports which serve as an extension of the on-chip I/O and are addressed as ports 4-7. The following operations may be performed on these ports:

- Transfer Accumulator to Port.
- Transfer Port to Accumulator.
- AND Accumulator to Port.
- OR Accumulator to Port.

All communication between the 8048 and the 8243 occurs over Port 2 (P20-P23) with timing provided by an output pulse on the PROG pin of the processor. Each transfer consists of two 4-bit nibbles:

The first containing the "op code" and port address and the second containing the actual 4-bits of data. A high to low transition of the PROG line indicates that address is present while a low to high transition indicates the presence of data. Additional 8243's may be added to the 4-bit bus and chip selected using additional output lines from the 8048/8748/8035.

Power On Initialization

Initial application of power to the device forces input/output ports 4, 5, 6, and 7 to the tri-state and port 2 to the input mode. The PROG pin may be either high or low when power is applied. The first high to low transition of PROG causes device to exit power on mode. The power on sequence is initiated if VCC drops below 1V.

P21	P20	Address Code	P23	P22	Instruction Code
0	0	Port 4	0	0	Read
0	1	Port 5	0	1	Write
1	0	Port 6	1	0	ORLD
1	1	Port 7	1	1	ANLD

Write Modes

The device has three write modes. MOVD Pi, A directly writes new data into the selected port and old data is lost. ORLD Pi, A takes new data, OR's it with the old data and then writes it to the port. ANLD Pi, A takes new data, AND's it with the old data and then writes it to the port. Operation code and port address are latched from the input port 2 on the high to low transition of the PROG pin. On the low to high transition of PROG data on port 2 is transferred to the logic block of the specified output port.

After the logic manipulation is performed, the data is latched and outputed. The old data remains latched until new valid outputs are entered.

Read Mode

The device has one read mode. The operation code and port address are latched from the input port 2 on the high to low transition of the PROG pin. As soon as the read operation and port address are decoded, the appropriate outputs are tri-stated, and the input buffers switched on. The read operation is terminated by a low to high transition of the PROG pin. The port (4, 5, 6 or 7) that was selected is switched to the tri-stated mode while port 2 is returned to the input mode.

Normally, a port will be in an output (write mode) or input (read mode). If modes are changed during operation, the first read following a write should be ignored; all following reads are valid. This is to allow the external driver on the port to settle after the first read instruction removes the low impedance drive from the 8243 output. A read of any port will leave that port in a high impedance state.

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias ..... 0°C to 70°C
Storage Temperature ..... -65°C to +150°C
Voltage on Any Pin
With Respect to Ground ..... -0.5 V to +7V
Power Dissipation ..... 1 Watt

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

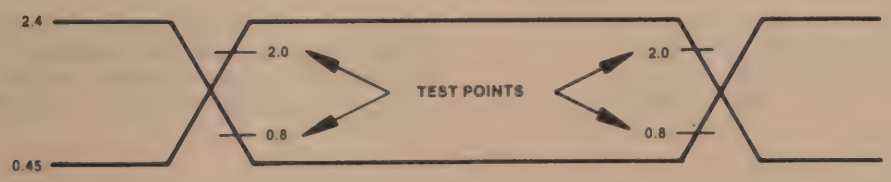
D.C. CHARACTERISTICS T_A = 0°C to 70°C, V_{CC} = 5V 10%

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
V_{IL}	Input Low Voltage	-0.5		0.8	V	
V_{IH}	Input High Voltage	2.0		V_{CC}+0.5	V	
V_{OL1}	Output Low Voltage Ports 4-7			0.45	V	I_{OL} = 4.5 mA*
V_{OL2}	Output Low Voltage Port 7			1	V	I_{OL} = 20 mA
V_{OH1}	Output High Voltage Ports 4-7	2.4			V	I_{OH} = 240μA
I_{IL1}	Input Leakage Ports 4-7	-10		20	μA	V_{in} = V_{CC} to OV
I_{IL2}	Input Leakage Port 2, CS, PROG	-10		10	μA	V_{in} = V_{CC} to OV
V_{OL3}	Output Low Voltage Port 2			.45	V	I_{OL} = 0.6 mA
I_{CC}	V_{CC} Supply Current		10	20	mA	
V_{OH2}	Output Voltage Port 2	2.4				I_{OH} = 100μA
I_{OL}	Sum of all I_{OL} from 16 Outputs			72	mA	4.5 mA Each Pin

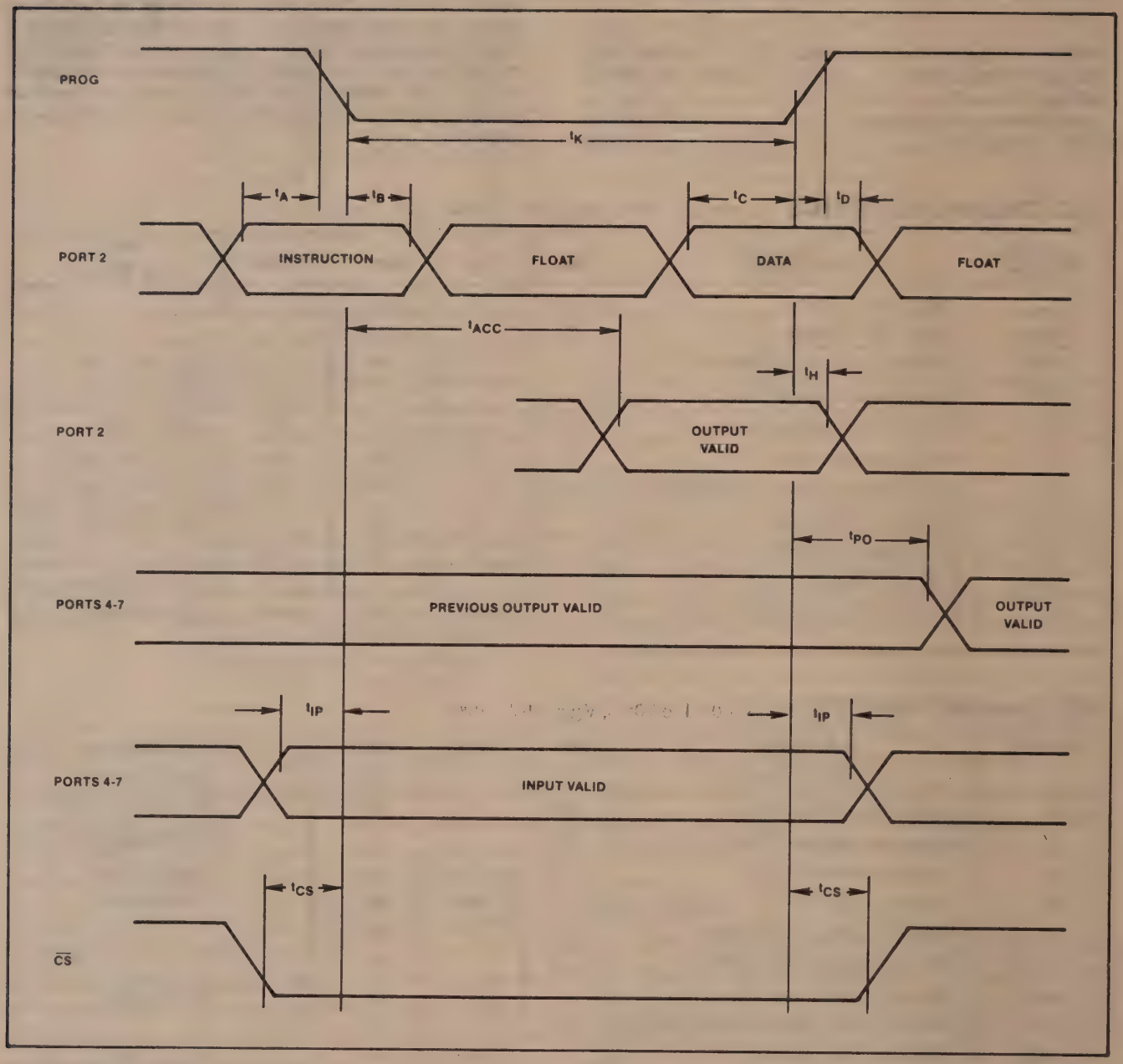
*See following graph for additional sink current capability

A.C. CHARACTERISTICS T_A = 0°C to 70°C, V_{CC} = 5V 10%

Symbol	Parameter	Min	Max	Units	Test Conditions
t_A	Code Valid Before PROG	100		ns	80 pF Load
t_B	Code Valid After PROG	60		ns	20 pF Load
t_C	Data Valid Before PROG	200		ns	80 pF Load
t_D	Data Valid After PROG	20		ns	20 pF Load
t_H	Floating After PROG	0	150	ns	20 pF Load
t_K	PROG Negative Pulse Width	700		ns	
t_{CS}	CS Valid Before/After PROG	50		ns	
t_{PO}	Ports 4-7 Valid After PROG		700	ns	100 pF Load
t_{LP1}	Ports 4-7 Valid Before/After PROG	100		ns	
t_{ACC}	Port 2 Valid After PROG		650	ns	80 pF Load



WAVEFORMS


PROG
t_K
t_A
t_B
t_C
t_D
PORT 2
INSTRUCTION
FLOAT
DATA
FLOAT
t_{ACC}
t_H
PORT 2
OUTPUT VALID
t_{PO}
PORTS 4-7
PREVIOUS OUTPUT VALID
OUTPUT VALID
t_{IP}
t_{IP}
PORTS 4-7
INPUT VALID
t_{CS}
t_{CS}
$\overline{CS}$

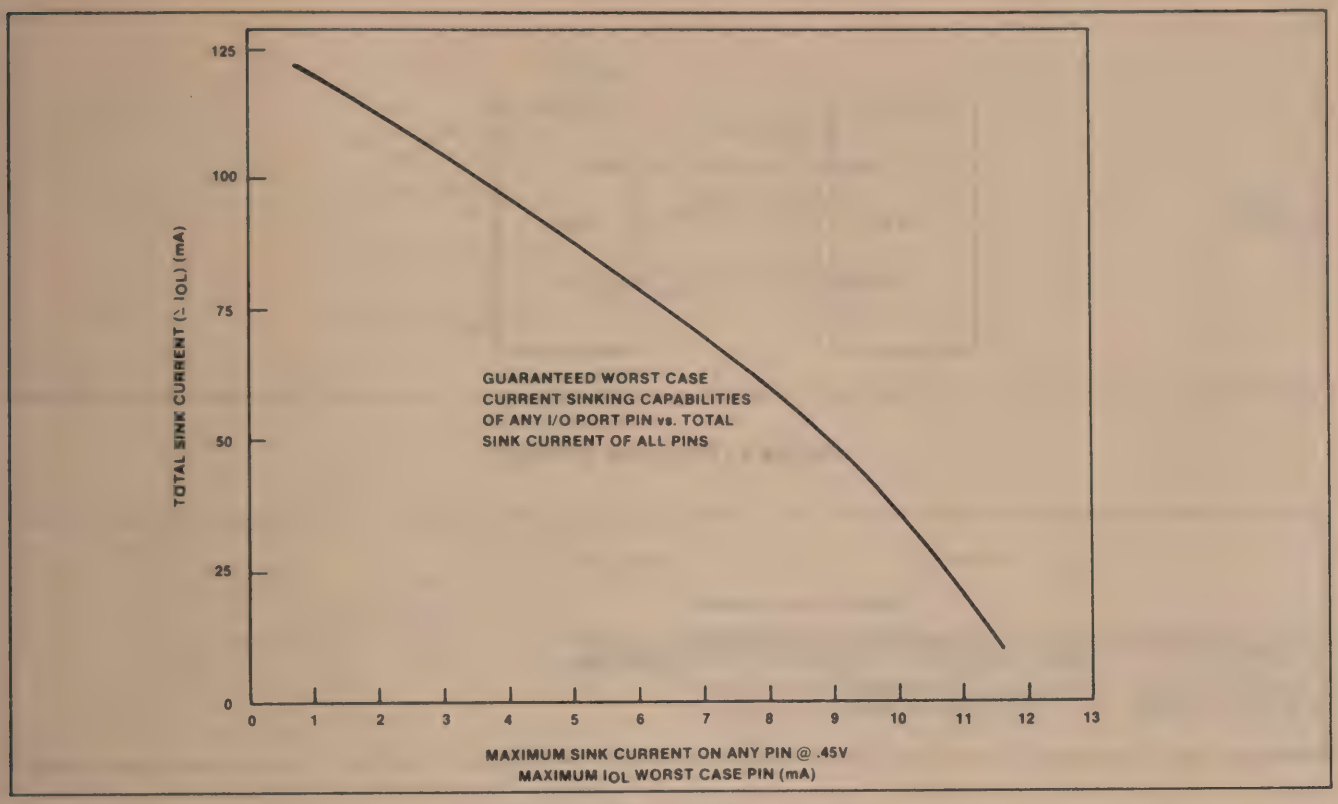


Figure 3

Sink Capability

The 8243 can sink 5 mA @ .45V on each of its 16 I/O lines simultaneously. If, however, all lines are not sinking simultaneously or all lines are not fully loaded, the drive capability of any individual line increases as is shown by the accompanying curve.

For example, if only 5 of the 16 lines are to sink current at one time, the curve shows that each of those 5 lines is capable of sinking 9 mA @ .45V (if any lines are to sink 9 mA the total IOL must not exceed 45 mA or five 9 mA loads).

Example: How many pins can drive 5 TTL loads (1.6 mA) assuming remaining pins are unloaded?

IOL = 5 x 1.6 mA = 8 mA
εIOL = 60 mA from curve
pins = 60 mA ÷ 8 mA/pin = 7.5 = 7

In this case, 7 lines can sink 8 mA for a total of 56mA. This leaves 4 mA sink current capability which can be divided in any way among the remaining 8 I/O lines of the 8243.

Example: This example shows how the use of the 20 mA sink capability of Port 7 affects the sinking capability of the other I/O lines.

An 8243 will drive the following loads simultaneously.

2 loads—20 mA @ 1V (port 7 only)
8 loads—4 mA @ .45V
6 loads—3.2 mA @ .45V
Is this within the specified limits?

εIOL = (2 x 20) + (8 x 4) + (6 x 3.2) = 91.2 mA.
From the curve: for IOL = 4 mA, εIOL ≈ 93 mA.
since 91.2 mA < 93 mA the loads are within specified limits.

Although the 20 mA @ 1V loads are used in calculating εIOL, it is the largest current required @ .45V which determines the maximum allowable εIOL.

NOTE: A10 to 50KΩ pullup resistor to +5V should be added to 8243 outputs when driving to 5V CMOS directly.

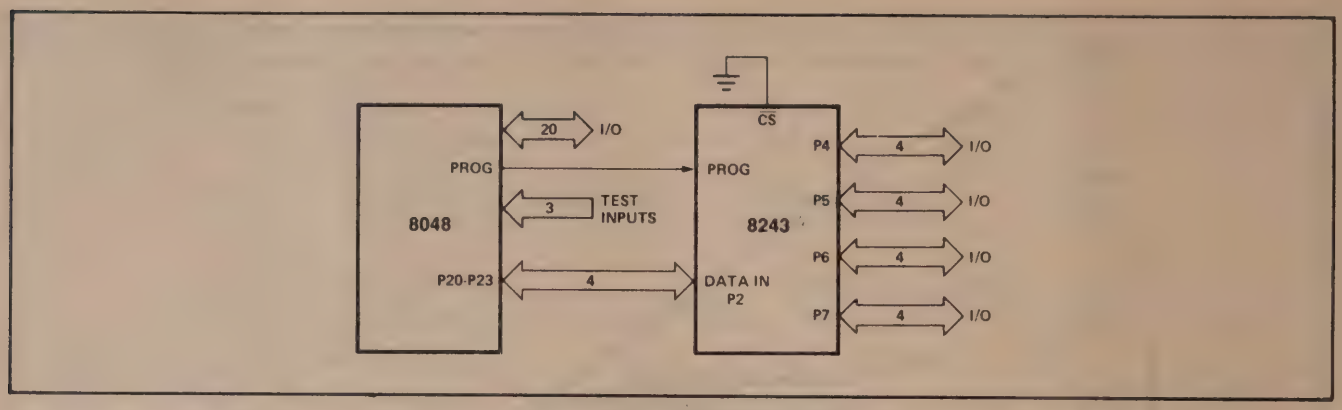


Figure 4. Expander Interface

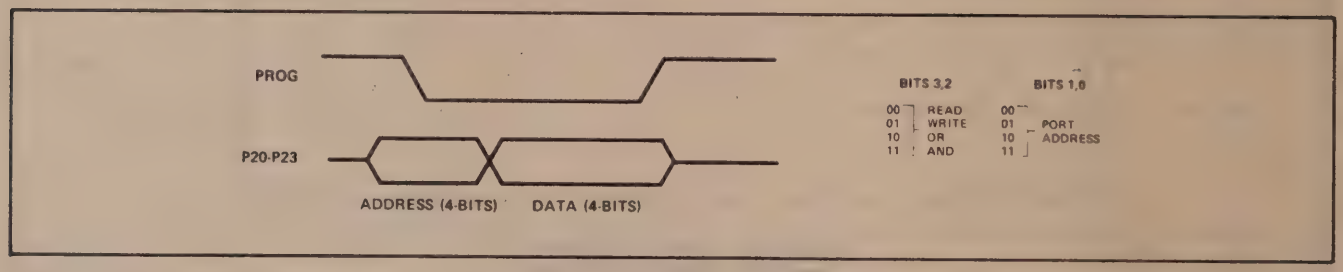


Figure 5. Output Expander Timing

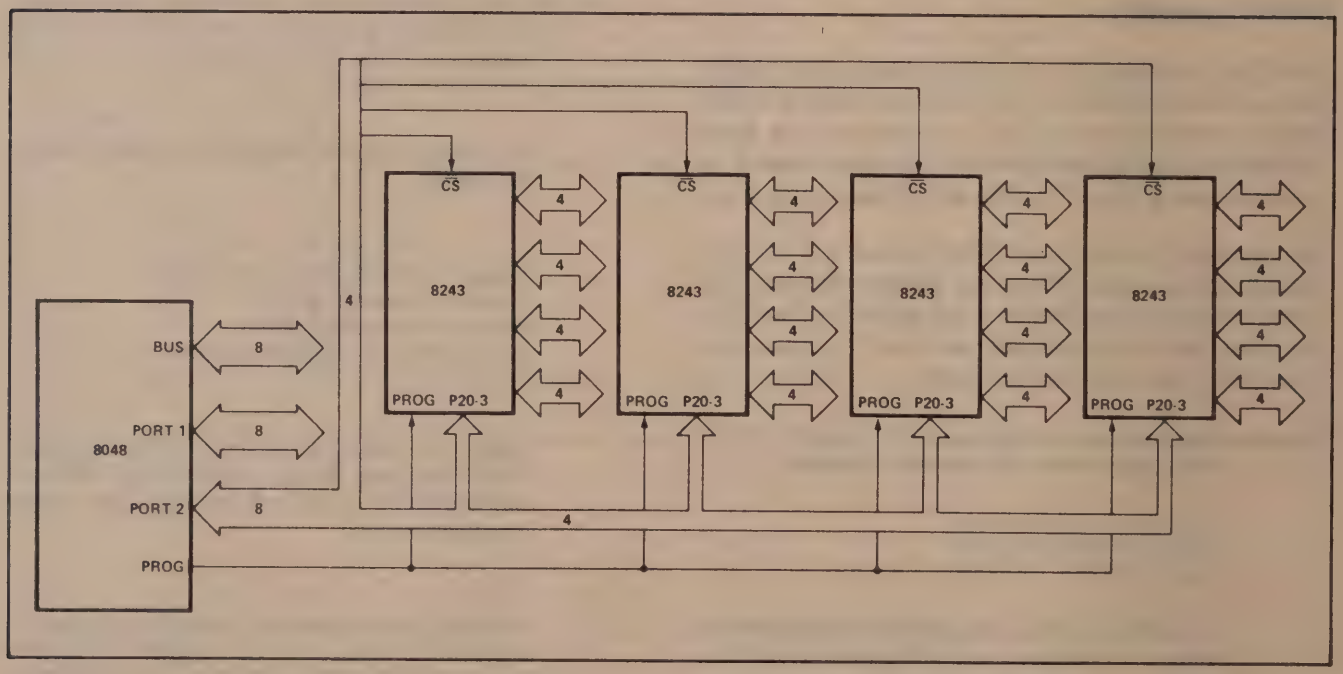


Figure 6. Using Multiple 8243's

intel

MICROPROCESSOR PERIPHERALS
UPI™ USER'S MANUAL

APRIL 1982

ORDER NUMBER: 231318-001

CHAPTER 1
INTRODUCTION

Accompanying the introduction of microprocessors such as the 8080, 8085, 8088, and 8086 there has been a rapid proliferation of intelligent peripheral devices. These special purpose peripherals extend CPU performance and flexibility in a number of important ways.

Table 1-1. Intelligent Peripheral Devices

8255 (GPIO)	Programmable Peripheral Interface
8251A (USART)	Programmable Communication Interface
8253 (TIMER)	Programmable Interval Timer
8257 (DMA)	Programmable DMA Controller
8259	Programmable Interrupt Controller
8271 (SDFDC), 8272 (DDFDC)	Programmable Floppy Disk Controllers
8273 (SDLC)	Programmable Synchronous Data Link Controller
8274	Programmable Multiprotocol-Serial Communications Controller
8275/8276 (CRT)	Programmable CRT Controllers
8279 (PKD)	Programmable Keyboard/Display Controller
8291A, 8292, 8293	Programmable GPIB System Talker, Listener, Controller

Intelligent devices like the 8272 floppy disk controller and 8273 synchronous data link controller (see Table 1-1) can preprocess serial data and perform control tasks which off-load the main system processor. Higher overall system throughput is achieved and software complexity is greatly reduced. The intelligent peripheral chips simplify master processor control tasks by performing many functions externally in peripheral hardware rather than internally in main processor software.

Intelligent peripherals also provide system flexibility. They contain on-chip mode registers which are programmed by the master processor during system initialization. These control registers allow the peripheral to be configured into many different operation modes. The user-defined program for the peripheral is stored in main system memory and is transferred to the peripheral's registers whenever a mode change is required. Of course, this type of flexibility requires software overhead in the master system which tends to limit the benefit derived from the peripheral chip.

In the past, intelligent peripherals were designed to handle very specialized tasks. Separate chips were designed for communication disciplines, parallel I/O, keyboard encoding, interval timing, CRT control, etc. Yet, in spite of the large number of devices available and the increased flexibility built into these chips, there is still a large number of microcomputer peripheral control tasks which are not satisfied.

With the introduction of the Universal Peripheral Interface (UPI) microcomputer, Intel has taken the intelligent peripheral concept a step further by providing an intelligent controller that is fully user programmable. It is a complete single-chip microcomputer which can connect directly to a master processor data bus. It has the same advantages of intelligence and flexibility which previous peripheral chips offered. In addition, the UPI is user-programmable: it has 1K bytes of ROM or EPROM memory for program storage plus 64 bytes of RAM memory for data storage or initialization from the master processor. The UPI device allows a designer to fully specify his control algorithm in the peripheral chip without relying on the master processor. Devices like printer controllers and keyboard scanners can be completely self-contained, relying on the master processor only for data transfer.

The UPI family currently consists of five components:

- 8741A microcomputer with 1K EPROM memory
- 8041AH microcomputer with 1K ROM memory
- 8042 microcomputer with 2K ROM memory
- 8243 I/O expander device
- 8742 microcomputer with 2K EPROM memory

The 8741A, 8041AH, 8742 and 8042 single chip microcomputers are functionally equivalent except for the type and amount of program memory available with each. These devices have the following main features:

- 8-bit CPU
- 8-bit data bus interface registers
- 1K by 8 bit ROM or EPROM memory (2K for 8042/8742)
- 64 by 8 bit RAM memory (128 bytes for 8042/8742)
- Interval timer/event counter
- Two 8-bit TTL compatible I/O ports
- Resident clock oscillator
- 12 MHZ operation, 1.25 μsec instruction cycle for 8041AH, 8742, 8042

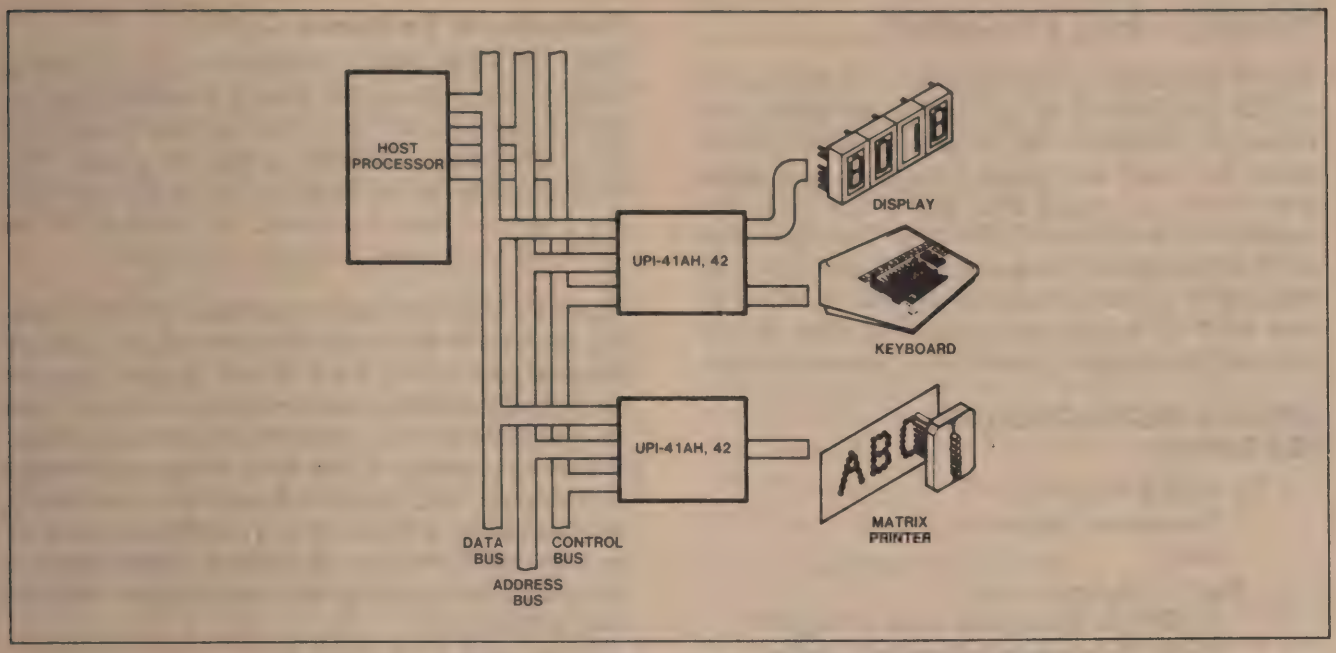


Figure 1-1. Interfacing Peripherals To Microcomputer Systems

HMOS processing has been applied to the UPI family to allow for additional performance and memory capability while reducing costs. The 8041AH, 8741A, 8042, 8742 are all pin and software compatible. This allows growth in present designs to incorporate new features and add additional performance. For new designs, the additional memory and performance of the 8042/8742 extends the UPI 'grow your own solution' concept to more complex motor control tasks, 80-column printers and process control applications as examples.

The 8243 device is an I/O multiplexer which allows expansion of I/O to over 100 lines (if seven devices are used). All three parts are fabricated with N-channel MOS technology and require a single, 5V supply for operation.

INTERFACE REGISTERS FOR MULTI-PROCESSOR CONFIGURATIONS

In the normal configuration, the 8041AH/8741A, 8042/8742 interfaces to the system bus, just like any intelligent peripheral device (see Figure 1-1). The host processor and the 8041AH/8741A, 8042/8742 form a loosely coupled multi-processor system, that is, communications between the two processors are direct. Common resources are three addressable registers located physically on the 8041AH/8741A, 8042/8742. These registers are the Data Bus Buffer Input (DBBIN), Data Bus Buffer Output (DBBOUT), and Status (STATUS) registers. The host processor may read data from DBBOUT or write commands and data into DBBIN. The status of DBBOUT and DBBIN plus user-defined status is supplied in STATUS. The host may read STATUS at any time. An interrupt to the UPI processor is automatically generated (if enabled) when DBBIN is loaded.

Because the UPI contains a complete microcomputer with program memory, data memory, and CPU it can function as a "Universal" controller. A designer can program the UPI to control printers, tape transports, or multiple serial communication channels. The UPI can also handle off-line arithmetic processing, or any number of other low speed control tasks.

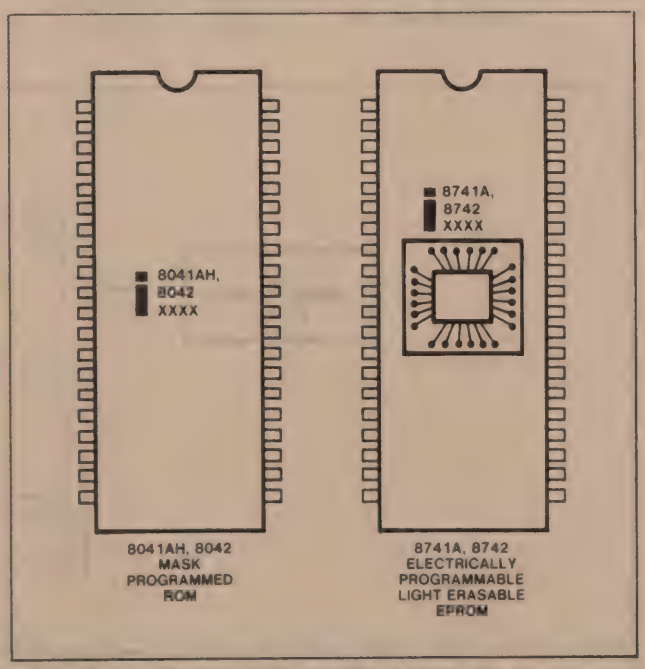


Figure 1-2. Pin Compatible ROM/EPROM Versions

POWERFUL 8-BIT PROCESSOR

The UPI contains a powerful, 8-bit CPU with as fast as 1.25 μsec cycle time and two single-level interrupts. Its instruction set includes over 90 instructions for easy software development. Most instructions are single byte and single cycle and none are more than two bytes long. The instruction set is optimized for bit manipulation and I/O operations. Special instructions are included to allow binary or BCD arithmetic operations, table lookup routines, loop counters, and N-way branch routines.

SPECIAL INSTRUCTION SET FEATURES

- For Loop Counters:
  Decrement Register and Jump if not zero.
- For Bit Manipulation:
  AND to A (immediate data or Register)
  OR to A (immediate data or Register)
  XOR to A (immediate data or Register)
  AND to Output Ports (Accumulator)
  OR to Output Ports (Accumulator)
  Jump Conditionally on any bit in A
- For BDC Arithmetic:
  Decimal Adjust A
  Swap 4-bit Nibbles of A
  Exchange lower nibbles of A and Register
  Rotate A left or right with or without Carry
- For Lookup Tables:
  Load A from Page of ROM (Address in A)
  Load A from Current Page of ROM (Address in A)

Features for Peripheral Control

The UPI 8-bit interval timer/event counter can be used to generate complex timing sequences for control applications or it can count external events such as switch closures and position encoder pulses. Software timing loops can be simplified or eliminated by the interval timer. If enabled, an interrupt to the CPU will occur when the timer overflows.

The UPI I/O complement contains two TTL-compatible 8-bit bidirectional I/O ports and two general-purpose test inputs. Each of the 16 port lines can individually function as either input or output under software control. Four of the port lines can also function as an interface for the 8243 I/O expander which provides four additional 4-bit ports that are directly addressable by UPI software. The 8243 expander allows low cost I/O expansion for large control applications while maintaining easy and efficient software port addressing.

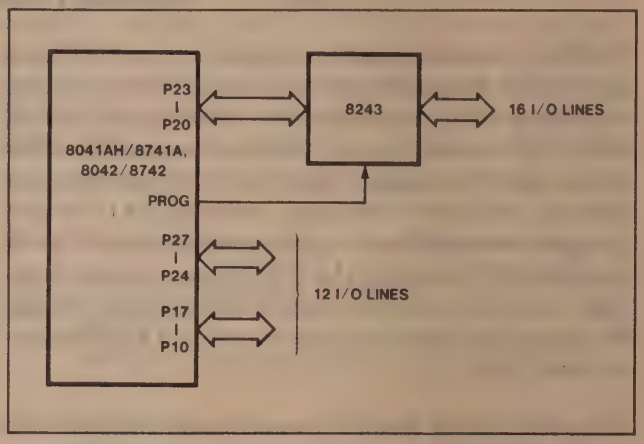


Figure 1-4. 8243 I/O Expander Interface

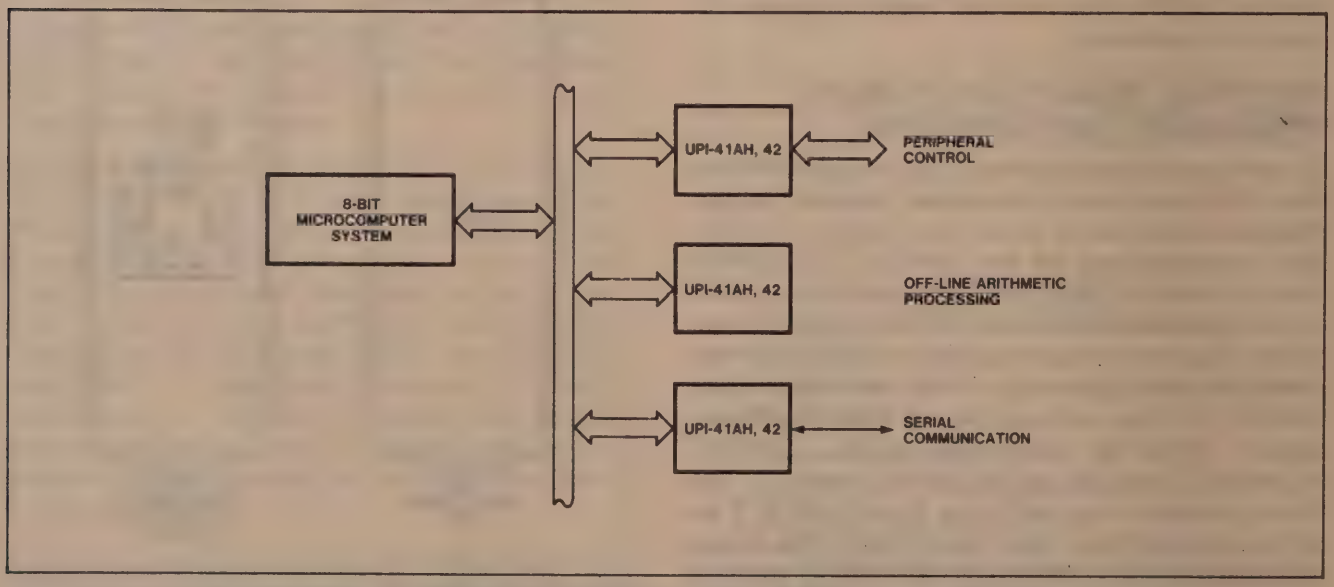


Figure 1-3. Interfaces And Protocols For Multiprocessor Systems

On-Chip Memory

The UPI's 64 (128) bytes of data memory include dual working register banks and an 8-level program counter stack. Switching between the register banks allows fast response to interrupts. The stack is used to store return addresses and processor status upon entering a subroutine.

The UPI program memory is available in two types to allow flexibility in moving from design to prototype to production with the same PC layout. The 8741A, 8742 device with EPROM memory is very economical for initial system design and development. Its program memory can be electrically programmed using the Intel Universal PROM Programmer. When changes are needed, the entire program can be erased using UV lamp and reprogrammed in about 20 minutes. This means the 8741A/8742 can be used as a single chip "breadboard" for very complex interface and control problems. After the 8741A/8742 is programmed it can be tested in the actual production level PC board and the actual functional environment. Changes required during system debugging can be made in the 8741A/8742 program much more easily than they could be made in a random logic design. The system configuration and PC layout can remain fixed during the development process and the turn around time between changes can be reduced to a minimum.

At any point during the development cycle, the 8741A/8742 EPROM part can be replaced with the low cost 8041AH, 8042 respectively with factory mask programmed memory. The transition from system development to mass production is made smoothly because the 8741A and 8041AH, 8742 and 8042 parts are completely pin compatible. 8742s or 8042s can be used in an 8041AH/8741 socket. This feature allows extensive testing with the EPROM part, even into initial shipments to customers. Yet, the transition to low-cost ROM is simplified to the point of being merely a package substitution.

PREPROGRAMMED UPI's

The 8292, 8294, and 8295 are 8041A's that are programmed by Intel and sold as standard peripherals. The 8292 is a GPIB controller, part of a three chip GPIB system. The 8294 is a Data Encryption Unit that implements the National Bureau of Standards data encryption algorithm. The 8295 is a dot matrix printer controller designed especially for the LRC 7040 series dot matrix impact printers. These parts illustrate the great flexibility offered by the UPI family.

DEVELOPMENT SUPPORT

The UPI microcomputer is fully supported by Intel with development tools like the UPP PROM programmer already mentioned. An ICE-41A in-circuit emulator is also available to allow UPI software and hardware to be developed easily and quickly. The combination of device features and Intel development support make the UPI an ideal component for low-speed peripheral control applications.

UPI DEVELOPMENT SUPPORT

- 8048/8041AH/8042 Assembler
- Universal PROM Programmer UPP Series
- ICE-41A Module
- MULTI-ICE
- Insite User's Library
- Application Engineers
- Training Courses

CHAPTER 2
FUNCTIONAL DESCRIPTION

The UPI-41AH, 42 microcomputer is an intelligent peripheral controller designed to operate in iAPX-86, 88, MCS-85, MCS-80, MCS-51 and MCS-48 systems. The UPI'S architecture, illustrated in Figure 2-1, is based on a low cost, single-chip microcomputer with program memory, data memory, CPU, I/O, event timer and clock oscillator in a single 40-pin package. Special interface registers are included which enable the UPI to function as a peripheral to an 8-bit master processor.

This chapter provides a basic description of the UPI microcomputer and its system interface registers. Unless otherwise noted the descriptions in this section apply to both the 8741A, 8742 (with UV erasable program memory) and the 8041AH, 8042 (with factory mask programmed memory). These two devices are so similar that they can be considered identical under most circumstances. All functions described in this chapter apply to the 8041AH, 8042, and 8741A, 8742.

PIN DESCRIPTION

The 8041AH/8741A, 8042/8742 are packaged in 40-pin Dual In-Line (DIP) packages. The pin configuration for both devices is shown in Figure 2-2. Figure 2-3 illustrates the UPI Logic Symbol.

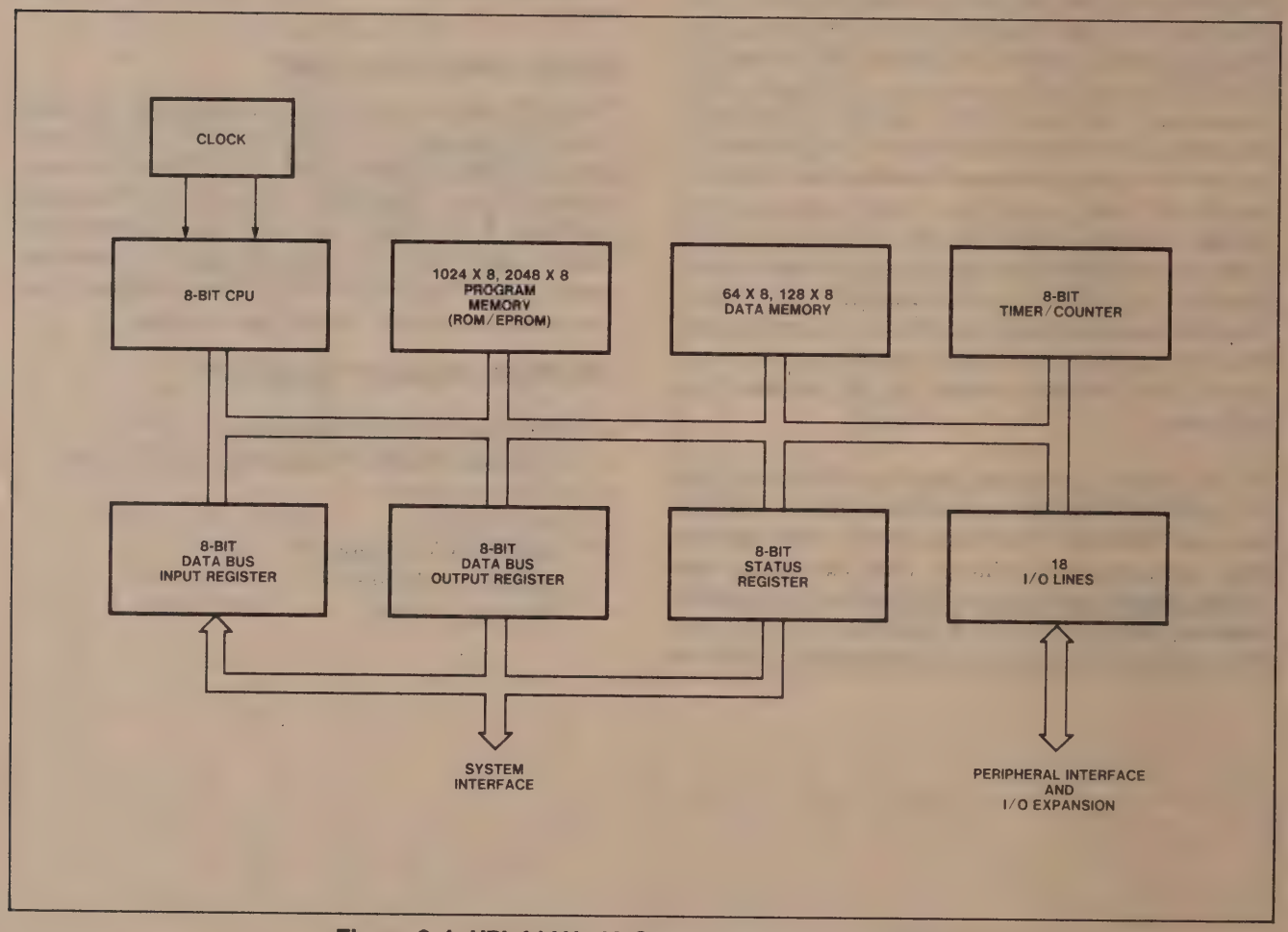


Figure 2-1. UPI-41AH, 42 Single Chip Microcomputer

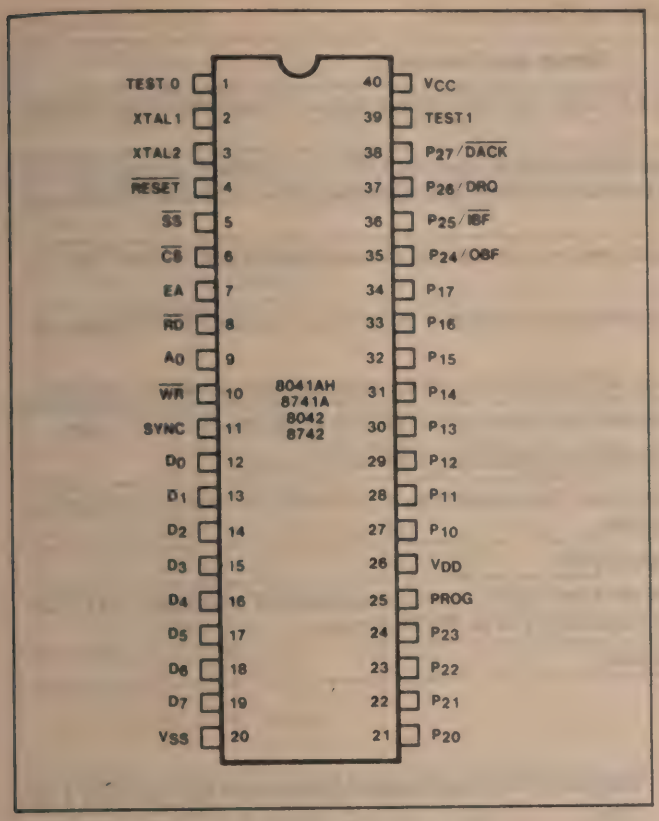


Figure 2-2. Pin Configuration

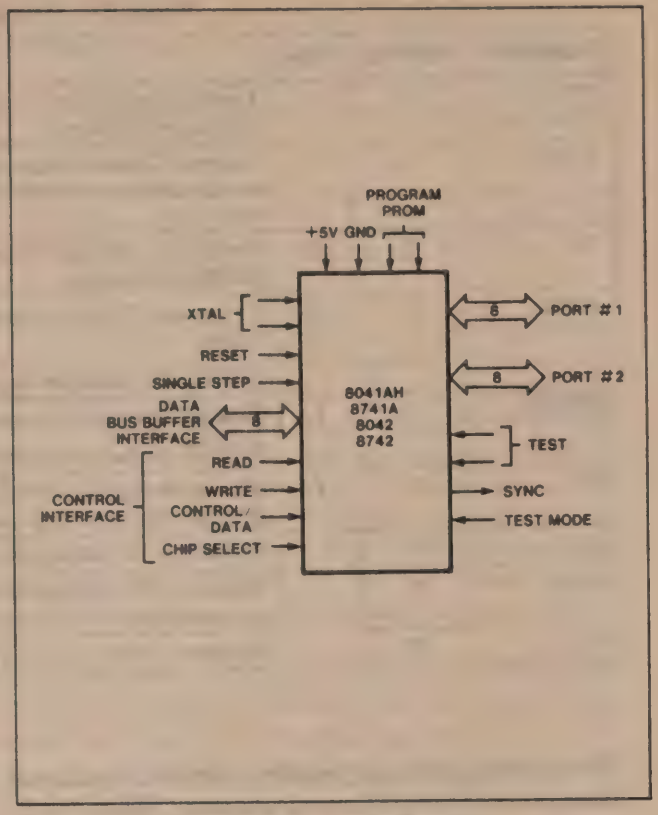


Figure 2-3. Logic Symbol

The following section summarizes the functions of each UPI-41A pin. NOTE that several pins have two or more functions which are described in separate paragraphs.

Table 2-1. Pin Description

Symbol	Pin No.	Type	Name and Function
D_0-D_7 (BUS)	12-19	I/O	**Data Bus:** Three-state, bidirectional DATA BUS BUFFER lines used to interface the UPI-41AH, 42 microcomputer to an 8-bit master system data bus.
P_{10}-P_{17}	27-34	I/O	**Port 1:** 8-bit, PORT 1 quasi-bidirectional I/O lines.
P_{20}-P_{27}	21-24 35-38	I/O	**Port 2:** 8-bit, PORT 2 quasi-bidirectional I/O lines. The lower 4 bits (P_{20}-P_{23}) interface directly to the 8243 I/O expander device and contain address and data information during PORT 4-7 access. The upper 4 bits (P_{24}-P_{27}) can be programmed to provide interrupt Request and DMA Handshake capability. Software control can configure P_{24} as Output Buffer Full (OBF) interrupt, P_{25} as Input Buffer Full ($\overline{IBF}$) interrupt, P_{26} as DMA Request (DRQ), and P_{27} as DMA ACKnowledge ($\overline{DACK}$).
$\overline{WR}$	10	I	**Write:** I/O write input which enables the master CPU to write data and command words to the UPI-41A INPUT DATA BUS BUFFER.
$\overline{RD}$	8	I	**Read:** I/O read input which enables the master CPU to read data and status words from the OUTPUT DATA BUS BUFFER or status register.
$\overline{CS}$	6	I	**Chip Select:** Chip select input used to select one UPI-41AH, 42 microcomputer out of several connected to a common data bus.
A_0	9	I	**Command/Data Select:** Address input used by the master processor to indicate whether byte transfer is data (A_0=0) or command (A_0=1).
TEST 0, TEST 1	1 39	I	**Test Inputs:** Input pins which can be directly tested using conditional branch instructions. **Frequency Reference:** TEST 1 (T_1) also functions as the event timer input (under software control). TEST 0 (T_0) is used during PROM programming and verification in the 8741A, 8742.

Table 2-1. Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
XTAL 1, XTAL 2	2 3	I	**Inputs:** Inputs for a crystal, LC or an external timing signal to determine the internal oscillator frequency.
SYNC	11	O	**Output Clock:** Output signal which occurs once per UPI-41A instruction cycle. SYNC can be used as a strobe for external circuitry; it is also used to synchronize single step operation.
EA	7	I	**External Access:** External access input which allows emulation, testing and PROM/ROM verification.
PROG	25	I/O	**Program:** Multifunction pin used as the program pulse input during PROM programming. During I/O expander access the PROG pin acts as an address/data strobe to the 8243.
$\overline{RESET}$	4	I	**Reset:** Input used to reset status flip-flops and to set the program counter to zero. $\overline{RESET}$ is also used during PROM programming and verification.
$\overline{SS}$	5	I	**Single Step:** Single step input used in conjunction with the SYNC output to step the program through each instruction.
V_{CC}	40		**Power:** +5V main power supply pin.
V_{DD}	26		**Power:** +5V during normal operation. +25V during programming operation, +21V for programming 8742. Low power standby pin in ROM version.
V_{SS}	20		**Ground:** Circuit ground potential.

The following sections provide a detailed functional description of the UPI microcomputer. Figure 2-4 illustrates the functional blocks within the UPI device.

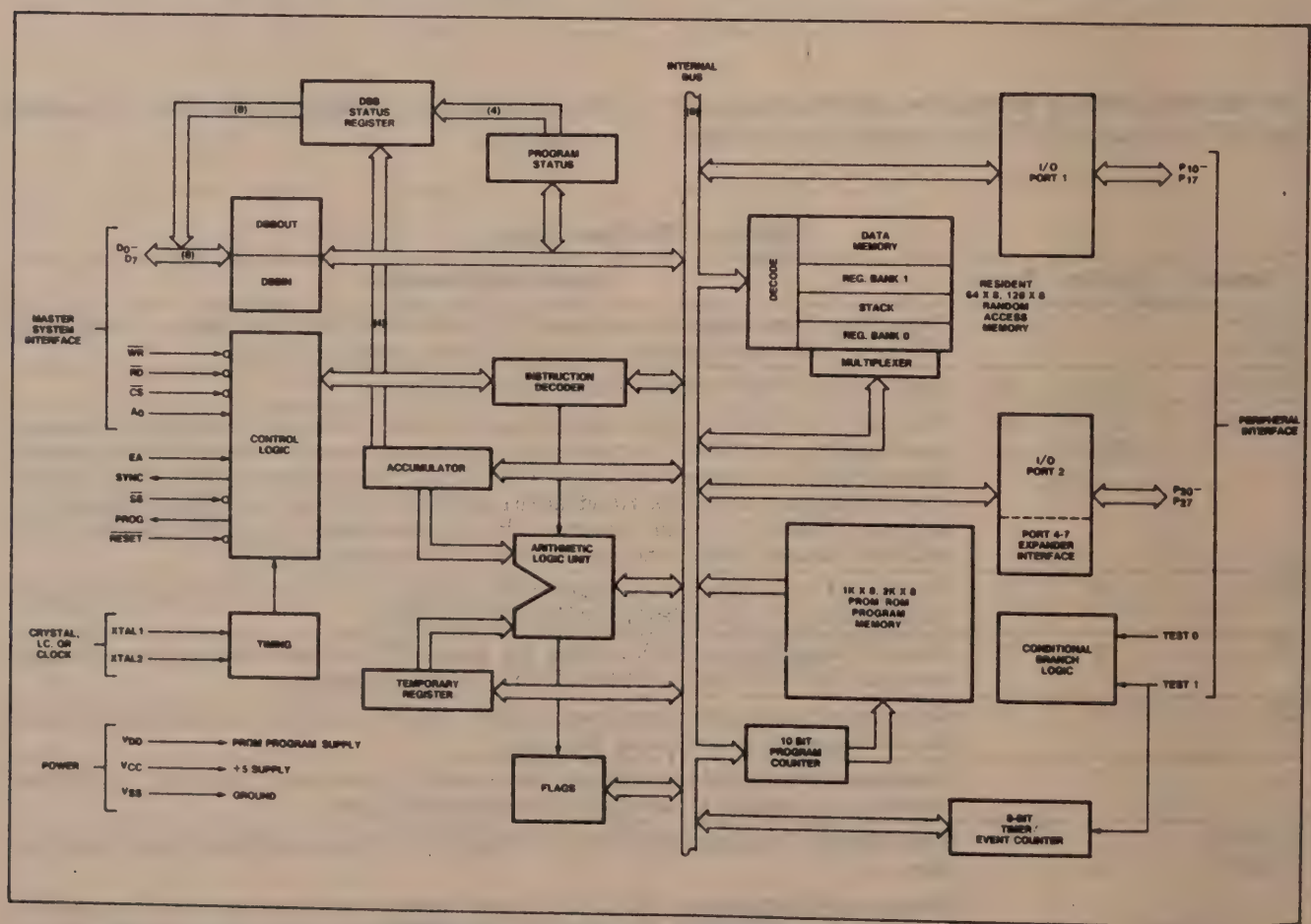


Figure 2-4. UPI-41AH, 42™ Block Diagram

CPU SECTION

The CPU section of the UPI-41AH, 42 microcomputer performs basic data manipulations and controls data flow throughout the single chip computer via the internal 8-bit data bus. The CPU section includes the following functional blocks shown in Figure 2-4:

- Arithmetic Logic Unit (ALU)
- Instruction Decoder
- Accumulator
- Flags

Arithmetic Logic Units (ALU)

The ALU is capable of performing the following operations:

- ADD with or without carry
- AND, OR, and EXCLUSIVE OR
- Increment, Decrement
- Bit complement
- Rotate left or right
- Swap
- BCD decimal adjust

In a typical operation data from the accumulator is combined in the ALU with data from some other source on the UPI-41AH, 42 internal bus (such as a register or an I/O port). The result of an ALU operation can be transferred to the internal bus or back to the accumulator.

If an operation such as an ADD or ROTATE requires more than 8 bits, the CARRY flag is used as an indicator. Likewise, during decimal adjust and other BCD operations the AUXILIARY CARRY flag can be set and acted upon. These flags are part of the Program Status Word (PSW).

Instruction Decoder

During an instruction fetch, the operation code (opcode) portion of each program instruction is stored and decoded by the instruction decoder. The decoder generates outputs used along with various timing signals to control the functions performed in the ALU. Also, the instruction decoder controls the source and destination of ALU data.

Accumulator

The accumulator is the single most important register in the processor. It is the primary source of data to the ALU and is often the destination for results as well. Data to and from the I/O ports and memory normally passes through the accumulator.

PROGRAM MEMORY

The UPI-41AH, 42 microcomputer has 1024, 2048 8-bit words of resident, read-only memory for program storage. Each of these memory locations is directly addressable by a 10-bit program counter. Depending on the type of application and the number of program changes anticipated, two types of program memory are available:

- 8041AH, 8042 with mask programmed ROM Memory
- 8741A, 8742 with electrically programmable EPROM Memory

The 8041AH and 8741A, 8042 and 8742 are functionally identical parts and are completely pin compatible. The 8742 and 8042 can be used in 8041AH, 8741A sockets. The 8041AH, 8042 has ROM memory which is mask programmed to user specification during fabrication. The 8741A/8742 are electrically programmed by the user using the Universal PROM Programmer (UPP series) with a UPP-848 or UPP-549 Personality Card. It can be erased using ultraviolet light and reprogrammed at any time.

A program memory map is illustrated in Figure 2-5. Memory is divided into 256 location 'pages' and three locations are reserved for special use:

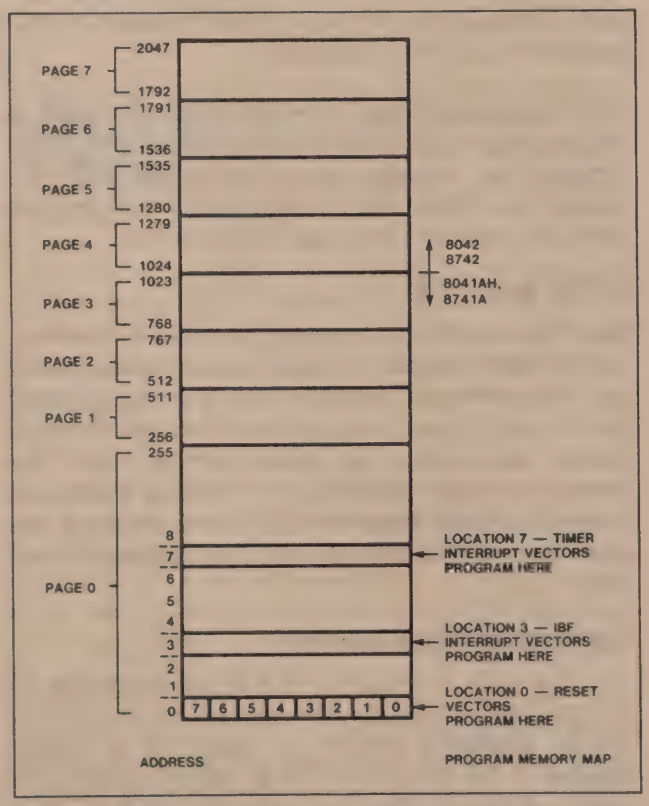


Figure 2-5. Program Memory Map

INTERRUPT VECTORS

1) **Location 0**
Following a $\overline{\text{RESET}}$ input to the processor, the next instruction is automatically fetched from location 0.

2) **Location 3**
An interrupt generated by an Input Buffer Full (IBF) condition (when the IBF interrupt is enabled) causes the next instruction to be fetched from location 3.

3) **Location 7**
A timer overflow interrupt (when enabled) will cause the next instruction to be fetched from location 7.

Following a system $\overline{\text{RESET}}$, program execution begins at location 0. Instructions in program memory are normally executed sequentially. Program control can be transferred out of the main line of code by an input buffer full (IBF) interrupt or a timer interrupt, or when a jump or call instruction is encountered. An IBF interrupt (if enabled) will automatically transfer control to location 3 while a timer interrupt will transfer control to location 7.

All conditional JUMP instructions and the indirect JUMP instruction are limited in range to the current 256-location page (that is, they alter PC bits 0–7 only). If a conditional JUMP or indirect JUMP begins in location 255 of a page, it must reference a destination on the following page.

Program memory can be used to store constants as well as program instructions. The UPI-41AH, 42 instruction set contains an instruction (MOVP3) designed specifically for efficient transfer of look-up table information from page 3 of memory.

DATA MEMORY

The UPI-41AH, 42 universal peripheral interface has 64, 128 8-bit words of random access data memory. This memory contains two working register banks, an 8-level program counter stack and a scratch pad memory, as shown in Figure 2-6. The amount of scratch pad memory available is variable depending on the number of addresses nested in the stack and the number of working registers being used.

Addressing Data Memory

The first eight locations in RAM are designated as working registers R_0-R_7. These locations (or registers) can be addressed directly by specifying a register number in the instruction. Since these locations are easily addressed, they are generally used to store frequently accessed intermediate results. Other locations in data memory are addressed indirectly by using R_0 or R_1 to specify the desired address. Since all RAM locations (including the eight working registers) can be addressed by 6 bits (UPI-41AH), and/or 7 bits (UPI-42), the most significant bit(s) of the address (6 and 7, or 7 only) are ignored.

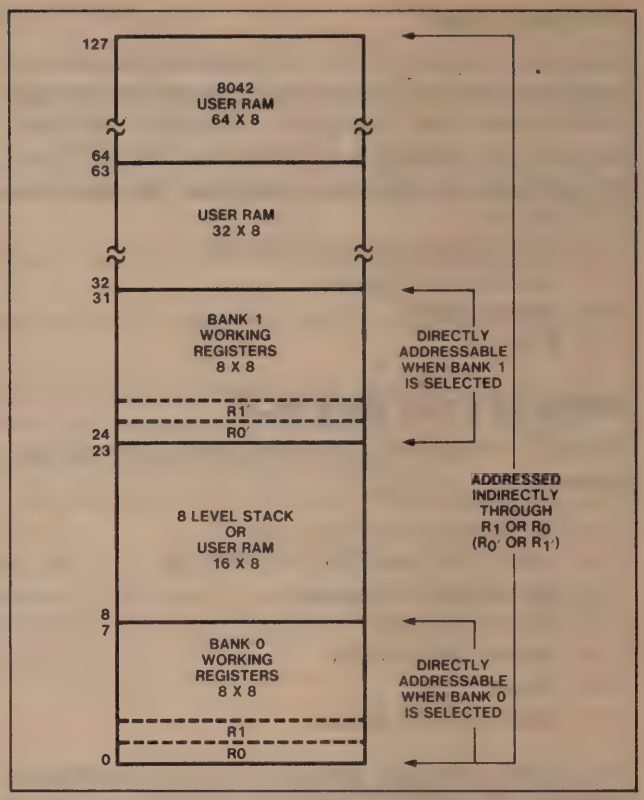


Figure 2-6. Data Memory Map

Working Registers

Dual banks of eight working registers are included in the UPI-41AH, 42 data memory. Locations 0–7 make up register bank 0 and locations 24–31 form register bank 1. A $\overline{\text{RESET}}$ signal automatically selects register bank 0. When bank 0 is selected, references to R_0–R_7 in UPI-41AH, 42 instructions operate on locations 0–7 in data memory. A "select register bank" instruction is used to select between the banks during program execution. If the instruction SEL RB1 (Select Register Bank 1) is executed, then program references to R_0–R_7 will operate on locations 24–31. As stated previously, registers 0 and 1 in the active register bank are used as indirect address registers for all locations in data memory.

Register bank 1 is normally reserved for handling interrupt service routines, thereby preserving the contents of the main program registers. The SEL RB1 instruction can be issued at the beginning of an interrupt service routine. Then, upon return to the main program, an RETR (return & restore status) instruction will automatically restore the previously selected bank. During interrupt processing, registers in bank 0 can be accessed indirectly using R_0' and R_1'.

If register bank 1 is not used, registers 24–31 can still serve as additional scratch pad memory.

Program Counter Stack

RAM locations 8–23 are used as an 8-level program counter stack. When program control is temporarily passed from the main program to a subroutine or interrupt service routine, the 10-bit program counter and bits 4–7 of the program status word (PSW) are stored in two stack locations. When control is returned to the main program via an RETR instruction, the program counter and PSW bits 4–7 are restored. Returning via an RET instruction does not restore the PSW bits, however. The program counter stack is addressed by three stack pointer bits in the PSW (bits 0–2). Operation of the program counter stack and the program status word is explained in detail in the following sections.

The stack allows up to eight levels of subroutine 'nesting'; that is, a subroutine may call a second subroutine, which may call a third, etc., up to eight levels. Unused stack locations can be used as scratch pad memory. Each unused level of subroutine nesting provides two additional RAM locations for general use.

The following sections provide a detailed description of the Program Counter Stack and the Program Status Word.

PROGRAM COUNTER

The UPI-41AH, 42 microcomputer has a 10-bit program counter (PC) which can directly address any of the 1024, 2048 locations in program memory. The program counter always contains the address of the next instruction to be executed and is normally incremented sequentially for each instruction to be executed when each instruction fetches occurs.

When control is temporarily passed from the main program to a subroutine or an interrupt routine, however, the PC contents must be altered to point to the address of the desired routine. The stack is used to save the current PC contents so that, at the end of the routine, main program execution can continue. The program counter is initialized to zero by a $\overline{\text{RESET}}$ signal.

PROGRAM COUNTER STACK

The Program Counter Stack is composed of 16 locations in Data Memory as illustrated in Figure 2-7. These RAM locations (8 through 23) are used to store the 10-bit program counter and 4 bits of the program status word.

An interrupt or CALL to a subroutine causes the contents of the program counter to be stored in one of the 8 register pairs of the program counter stack.

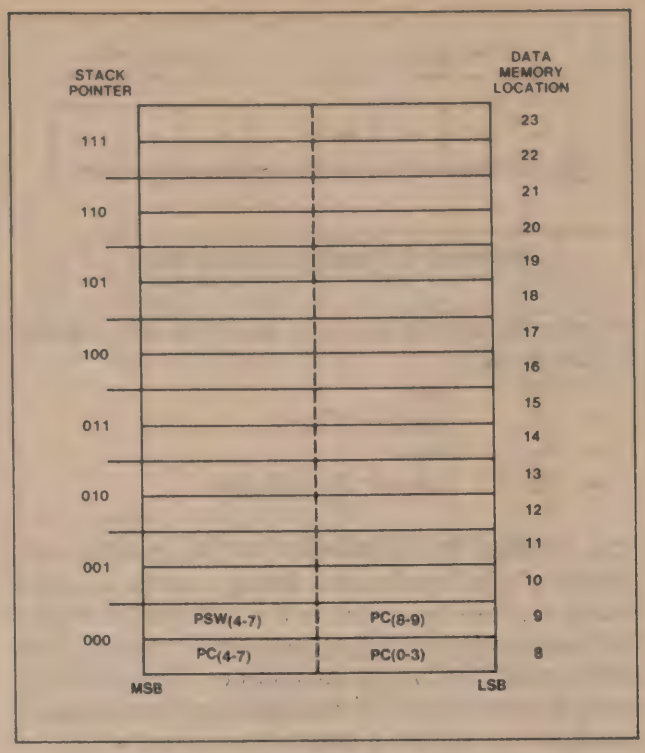


Figure 2-7. Program Counter Stack

A 3-bit Stack Pointer which is part of the Program Status Word (PSW) determines the stack pair to be used at a given time. The stack pointer is initialized by a $\overline{\text{RESET}}$ signal to 00H which corresponds to RAM locations 8 and 9.

The first call or interrupt results in the program counter and PSW contents being transferred to RAM locations 8 and 9 in the format shown in Figure 2-7. The stack pointer is automatically incremented by 1 to point to locations 10 and 11 in anticipation of another CALL.

Nesting of subroutines within subroutines can continue up to 8 levels without overflowing the stack. If overflow does occur the deepest address stored (locations 8 and 9) will be overwritten and lost since the stack pointer overflows from 07H to 00H. Likewise, the stack pointer will underflow from 00H to 07H.

The end of a subroutine is signaled by a return instruction, either RET or RETR. Each instruction will automatically decrement the Stack Pointer and transfer the contents of the proper RAM register pair to the Program Counter.

PROGRAM STATUS WORD

The 8-bit program status word illustrated in Figure 2-8 is used to store general information about program execution. In addition to the 3-bit Stack

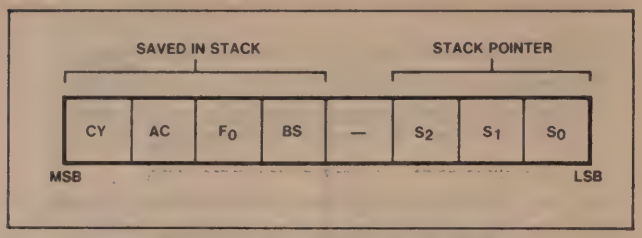


Figure 2-8. Program Status Word

Pointer discussed previously, the PSW includes the following flags:

- CY — Carry
- AC — Auxiliary Carry
- F_0 — Flag 0
- BS — Register Bank Select

The Program Status Word (PSW) is actually a collection of flip-flops located throughout the machine which are read or written as a whole. The PSW can be loaded to or from the accumulator by the MOV A, PSW or MOV PSW,A instructions. The ability to write directly to the PSW allows easy restoration of machine status after a power-down sequence.

The upper 4 bits of the PSW (bits 4, 5, 6, and 7) are stored in the PC Stack with every subroutine CALL or interrupt vector. Restoring the bits on a return is optional. The bits are restored if an RETR instruction is executed, but not if an RET is executed.

PSW bit definitions are as follows:

- Bits 0–2 Stack Pointer Bits S_0, S_1, S_2
- Bit 3 Not Used
- Bit 4 Working Register Bank
  0 = Bank 0
  1 = Bank 1
- Bit 5 Flag 0 bit (F_0)
  This is a general purpose flag which can be cleared or complemented and tested with conditional jump instructions. It may be used during data transfer to an external processor.
- Bit 6 Auxiliary Carry (AC)
  The flag status is determined by an ADD instruction and is used by the Decimal Adjustment instruction DAA.
- Bit 7 Carry (CY)
  The flag indicates that a previous operation resulted in overflow of the accumulator.

CONDITIONAL BRANCH LOGIC

Conditional Branch Logic in the UPI-41AH, 42 allows the status of various processor flags, inputs, and other hardware functions to directly affect program execution. The status is sampled in state 3 of the first cycle.

Table 2-2 lists the internal conditions which are testable and indicates the condition which will cause a jump. In all cases, the destination address must be within the page of program memory (256 locations) in which the jump instruction occurs.

OSCILLATOR AND TIMING CIRCUITS

The 8041A's internal timing generation is controlled by a self-contained oscillator and timing circuit. A choice of crystal, L-C or external clock can be used to derive the basic oscillator frequency.

The resident timing circuit consists of an oscillator, a state counter and a cycle counter as illustrated in Figure 2-9. Figure 2-10 shows instruction cycle timing.

Table 2-2. Conditional Branch Instructions

Device	Instruction Mnemonic		Jump Condition Jump if:
Accumulator	JZ	addr	All bits zero
	JNZ	addr	Any bit not zero
Accumulator bit	JBb	addr	Bit "b" = 1
Carry flag	JC	addr	Carry flag = 1
	JNC	addr	Carry flag = 0
User flag	JFO	addr	F_0 flag = 1
	JF1	addr	F_1 flag = 1
Timer flag	JTF	addr	Timer flag = 1
Test Input 0	JT0	addr	T_0 = 1
	JNT0	addr	T_0 = 0
Test Input 1	JT1	addr	T_1 = 1
	JNT1	addr	T_1 = 0
Input Buffer flag	JNIBF	addr	IBF flag = 0
Output Buffer flag	JOBF	addr	OBF flag = 1

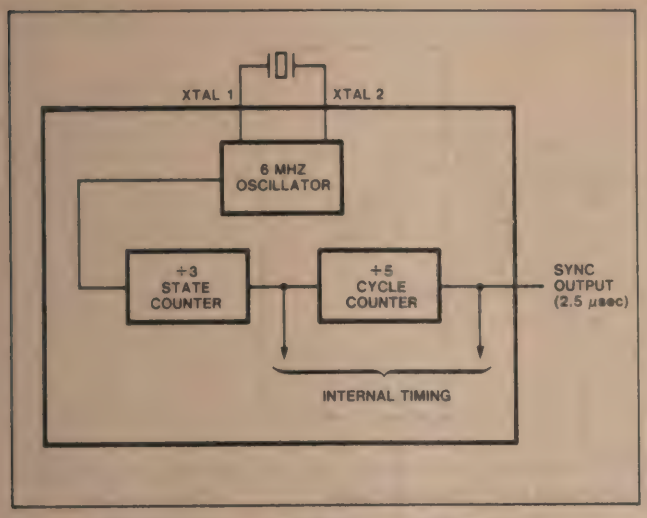


Figure 2-9. Oscillator Configuration

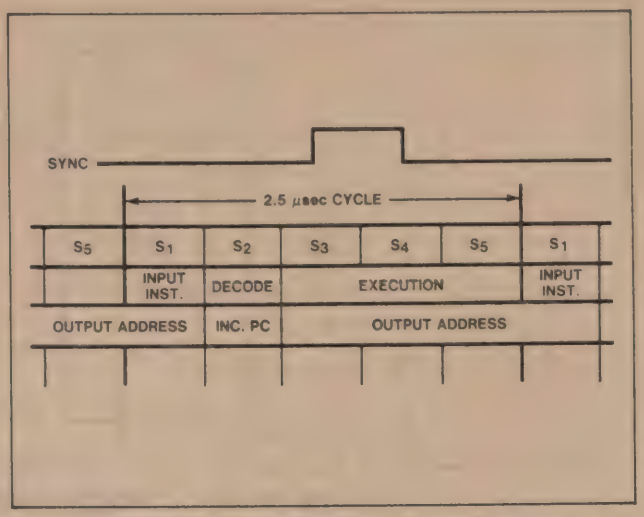


Figure 2-10. Instruction Cycle Timing

Oscillator

The on-board oscillator is a series resonant circuit with a frequency range of 1 to 12 (8041AH-2/8042/8742) MHz. Pins XTAL 1 and XTAL 2 are input and output (respectively) of a high gain amplifier stage. A crystal or inductor and capacitor connected between XTAL 1 and XTAL 2 provide the feedback and proper phase shift for oscillation. Recommended connections for crystal or L-C are shown in Figure 2-11.

State Counter

The output of the oscillator is divided by 3 in the state counter to generate a signal which defines the state times of the machine.

Each instruction cycle consists of five states as illustrated in Figure 2-10 and Table 2-3. The overlap of address and execution operations illustrated in Figure 2-10 allows fast instruction execution.

Table 2-3. Instruction Timing Diagram

INSTRUCTION	CYCLE 1					CYCLE 2				
	S1	S2	S3	S4	S5	S1	S2	S3	S4	S5
IN A,Pp	Fetch Instruction	Increment Program Counter	—	Increment Timer	—	—	Read Port	—	—	—
OUTL Pp,A	Fetch Instruction	Increment Program Counter	—	Increment Timer	Output To Port	—	—	—	—	—
ANL Pp, DATA	Fetch Instruction	Increment Program Counter	—	Increment Timer	Read Port	Fetch Immediate Data	—	Increment Program Counter	Output To Port	—
ORL Pp, DATA	Fetch Instruction	Increment Program Counter	—	Increment Timer	Read Port	Fetch Immediate Data	—	Increment Program Counter	Output To Port	—
MOVD A,Pp	Fetch Instruction	Increment Program Counter	Output Opcode/Address	Increment Timer	—	—	Read P2 Lower	—	—	—
MOVD Pp,A	Fetch Instruction	Increment Program Counter	Output Opcode/Address	Increment Timer	Output Data To P2 Lower	—	—	—	—	—
ANLD Pp,A	Fetch Instruction	Increment Program Counter	Output Opcode/Address	Increment Timer	Output Data	—	—	—	—	—
ORLD Pp,A	Fetch Instruction	Increment Program Counter	Output Opcode/Address	Increment Timer	Output Data	—	—	—	—	—
J (Conditional)	Fetch Instruction	Increment Program Counter	Sample Condition	Increment Timer	—	Fetch Immediate Data	—	Update Program Counter	—	—
MOV STS, A	Fetch Instruction	Increment Program Counter	—	Increment Timer	Update Status Register					
IN A,DBB	Fetch Instruction	Increment Program Counter	—	Increment Timer	—					
OUT DBB,A	Fetch Instruction	Increment Program Counter	—	Increment Timer	Output To Port					
STRT T STRT CNT	Fetch Instruction	Increment Program Counter	—	—	Start Counter					
STOP TCNT	Fetch Instruction	Increment Program Counter	—	—	Stop Counter					
EN I	Fetch Instruction	Increment Program Counter	—	Enable Interrupt	—					
DIS I	Fetch Instruction	Increment Program Counter	—	Disable Interrupt	—					
EN DMA	Fetch Instruction	Increment Program Counter	—	DMA Enabled DRQ Cleared	—					
EN FLAGS	Fetch Instruction	Increment Program Counter	—	$\overline{OBF}$ $\overline{IBF}$ Output Enabled	—					

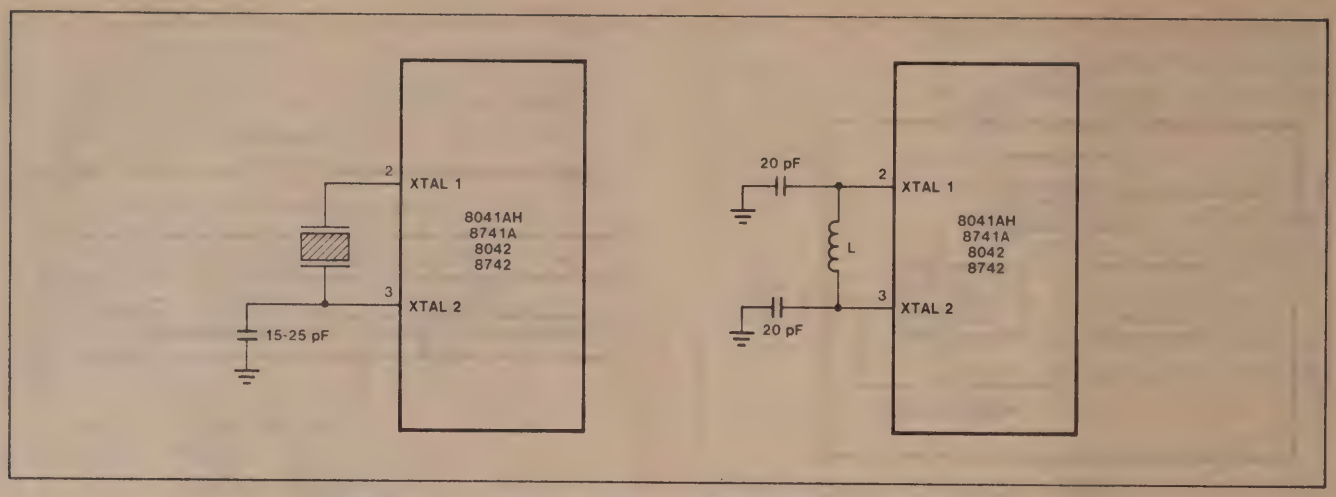


Figure 2-11. Recommended Crystal and L-C Connections

Cycle Counter

The output of the state counter is divided by 5 in the cycle counter to generate a signal which defines a machine cycle. This signal is call SYNC and is available continously on the SYNC output pin. It can be used to synchronize external circuitry or as a general purpose clock output. It is also used for synchronizing single-step.

Frequency Reference

The external crystal provides high speed and accurate timing generation. A crystal frequency of 5.9904 MHz is useful for generation of standard communication frequencies by the 8041AH/8741, 8042/8742. However, if an accurate frequency reference and maximum processor speed are not required, an inductor and capacitor may be used in place of the crystal as shown in Figure 2-11.

A recommended range of inductance and capacitance combinations is given below:

- L = 130 μH corresponds to 3 MHz
- L = 45 μH corresponds to 5 MHz

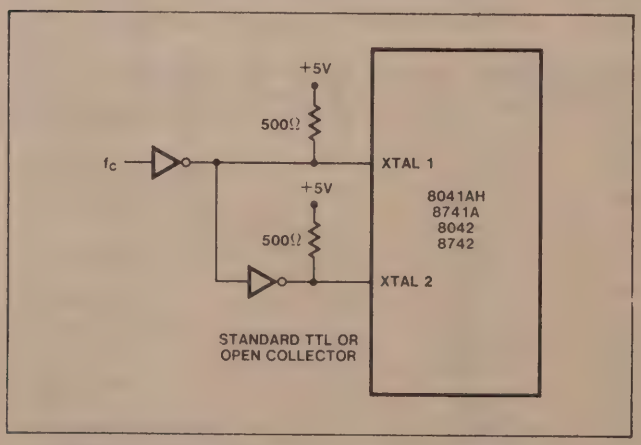


Figure 2-12. Recommended Connection For External Clock Signal

An external clock signal can also be used as a frequency reference to the 8741AH, 8741A, 8742 or 8042; however, the levels are *not* TTL compatible. The signal must be in the 1–12 MHz frequency range and must be connected to pins XTAL 1 and XTAL 2 by buffers with a suitable pull-up resistor to guarantee that a logic "1" is above 3.8 volts. The recommended connection is shown in Figure 2-12.

INTERVAL TIMER/EVENT COUNTER

The 8041AH, 8042 has a resident 8-bit timer/counter which has several software selectable modes of operation. As an interval timer, it can generate accurate delays from 80 microseconds to 20.48 milliseconds without placing undue burden on the processor. In the counter mode, external events such as switch closures or tachometer pulses can be counted and used to direct program flow.

Timer Configuration

Figure 2-13 illustrates the basic timer/counter configuration. An 8-bit register is used to count pulses from either the internal clock and prescaler or from an external source. The counter is presettable and readable with two MOV instructions which transfer the contents of the accumulator to the counter and vice-versa (i.e. MOV T, A and MOV A, T). The counter is stopped by a $\overline{\text{RESET}}$ or STOP TCNT instruction and remains stopped until restarted either as a timer (START T instruction) or as a counter (START CNT instruction). Once started, the counter will increment to its maximum count (FFH) and overflow to zero continuing its count until stopped by a STOP TCNT instruction or $\overline{\text{RESET}}$.

The increment from maximum count to zero (overflow) results in setting the Timer Flag (TF) and generating an interrupt request. The state of the overflow flag is testable with the conditional jump

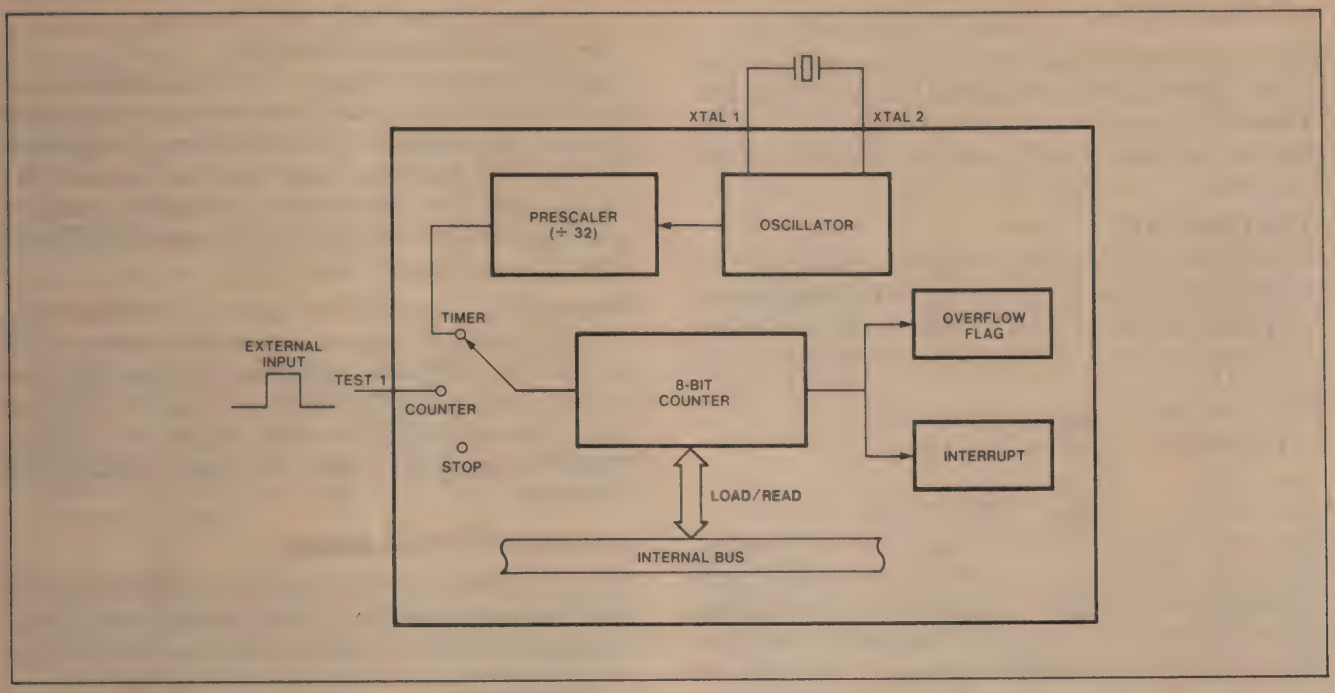


Figure 2-13. Timer Counter

instruction, JTF. The flag is reset by executing a JTF or by a $\overline{\text{RESET}}$ signal.

The timer interrupt request is stored in a latch and ORed with the input buffer full interrupt request. The timer interrupt can be enabled or disabled independent of the IBF interrupt by the EN TCNTI and DIS TCTNI instructions. If enabled, the counter overflow will cause a subroutine call to location 7 where the timer service routine is stored. If the timer and Input Buffer Full interrupts occur simultaneously, the IBF source will be recognized and the call will be to location 3. Since the timer interrupt is latched, it will remain pending until the DBBIN register has been serviced and will immediately be recognized upon return from the service routine. A pending timer interrupt is reset by the initiation of a timer interrupt service routine.

Event Counter Mode

The STRT CNT instruction connects the TEST 1 input pin to the counter input and enables the counter. Note this instruction does not clear the counter. The counter is incremented on high to low transitions of the TEST 1 input. The TEST 1 input must remain high for a minimum of one state in order to be registered (250 ns at 12 MHz). The maximum count frequency is one count per three instruction cycles (267 kHz at 12 MHz). There is no minimum frequency limit.

Timer Mode

The STRT T instruction connects the internal clock to the counter input and enables the counter. The input clock is derived from the SYNC signal of the internal oscillator and the divide-by-32 prescaler. The configuration is illustrated in Figure 2-13. Note this instruction does not clear the timer register. Various delays and timing sequences between 40 μsec and 10.24 msec can easily be generated with a minimum of software timing loops (at 12 MHz).

Times longer than 10.24 msec can be accurately measured by accumulating multiple overflows in a register under software control. For time resolution less than 40 μsec, an external clock can be applied to the TEST 1 counter input (see Event Counter Mode). The minimum time resolution with an external clock is 3.75 μsec (267 kHz at 12 MHz).

TEST 1 Event Counter Input

The TEST 1 pin is multifunctional. It is automatically initialized as a test input by a $\overline{\text{RESET}}$ signal and can be tested using UPI-41A conditional branch instructions.

In the second mode of operation, illustrated in Figure 2-13, the TEST 1 pin is used as an input to the internal 8-bit event counter. The Start Counter (STRT CNT) instruction controls an internal switch which connects TEST 1 through an edge detector to the 8-bit internal counter. Note that this instruction does not inhibit the testing of TEST 1 via conditional Jump instructions.

In the counter mode the TEST 1 input is sampled once per instruction cycle. After a high level is detected, the next occurence of a low level at TEST 1

will cause the counter to increment by one.

The event counter functions can be stopped by the Stop Timer/Counter (STOP TCNT) instruction. When this instruction is executed the TEST 1 pin becomes a test input and functions as previously described.

TEST INPUTS

There are two multifunction pins designated as Test Inputs, TEST 0 and TEST 1. In the normal mode of operation, status of each of these lines can be directly tested using the following conditional Jump instructions:

- JT0 Jump if TEST 0 = 1
- JNT0 Jump if TEST 0 = 0
- JT1 Jump if TEST 1 = 1
- JNT1 Jump if TEST 1 = 0

The test inputs are TTL compatible. An external logic signal connected to one of the test inputs will be sampled at the time the appropriate conditional jump instruction is executed. The path of program execution will be altered depending on the state of the external signal when sampled.

INTERRUPTS

The 8041AH/8741A, 8042/8742 has the following internal interrupts:

- Input Buffer Full (IBF) interrupt
- Timer Overflow interrupt

The IBF interrupt forces a CALL to location 3 in program memory; a timer-overflow interrupt forces a CALL to location 7. The IBF interrupt is enabled by the EN I instruction and disabled by the DIS I instruction. The timer-overflow interrupt is enabled and disabled by the EN TNCTI and DIS TCNTI instructions, respectively.

Figure 2-14 illustrates the internal interrupt logic. An IBF interrupt request is generated whenever $\overline{WR}$ and $\overline{CS}$ are both low, regardless of whether interrupts are enabled. The interrupt request is cleared upon entering the IBF service routine only. That is, the DIS I instruction does not clear a pending IBF interrupt.

Interrupt Timing Latency

When the IBF interrupt is enabled and an IBF interrupt request occurs, an interrupt sequence is initiated as soon as the currently executing instruction is completed. The following sequence occurs:

- A CALL to location 3 is forced.
- The program counter and bits 4–7 of the Program Status Word are stored in the stack.
- The stack pointer is incremented.

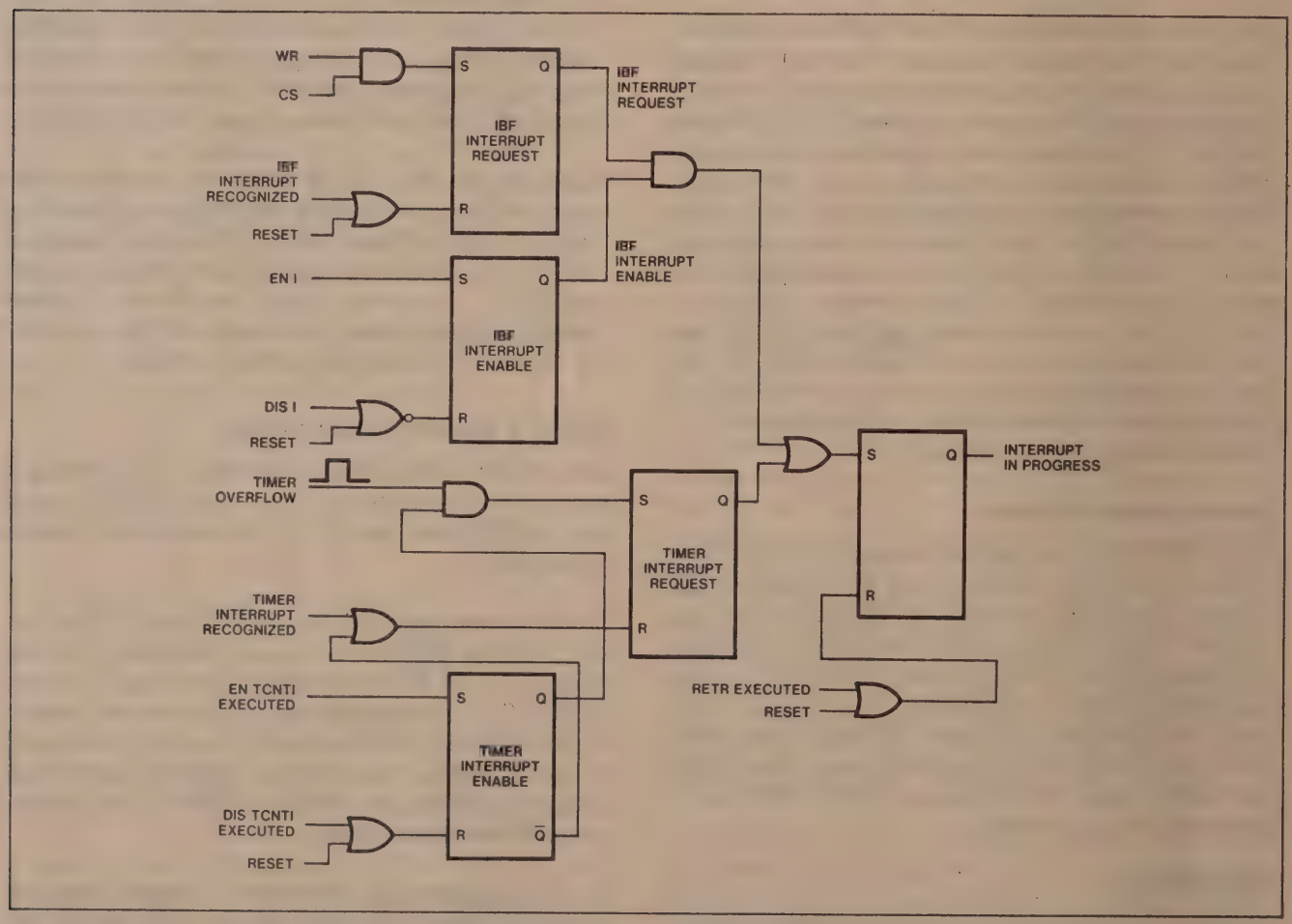


Figure 2-14. Interrupt Logic

Location 3 in program memory should contain an unconditional jump to the beginning of the IBF interrupt service routine elsewhere in program memory. At the end of the service routine, an RETR (Return and Restore Status) instruction is used to return control to the main program. This instruction will restore the program counter and PSW bits 4–7, providing automatic restoration of the previously active register bank as well. RETR also re-enables interrupts.

A timer-overflow interrupt is enabled by the EN TCNTI instruction and disabled by the DIS TCNTI instruction. If enabled, this interrupt occurs when the timer/counter register overflows. A CALL to location 7 is forced and the interrupt routine proceeds as described above.

The interrupt service latency is the sum of current instruction time, interrupt recognition time, and the internal call to the interrupt vector address. The worst case latency time for servicing an interrupt is 7 clock cycles. Best case latency is 4 clock cycles.

Interrupt Timing

Interrupt inputs may be enabled or disabled under program control using EN I, DIS I, EN TCNTI and DIS TCNTI instructions. Also, a $\overline{\text{RESET}}$ input will disable interrupts. An interrupt request must be removed before the RETR instruction is executed to return from the service routine, otherwise the processor will re-enter the service routine immediately. Thus, the $\overline{\text{WR}}$ and $\overline{\text{CS}}$ inputs should not be held low longer than the duration of the interrupt service routine.

The interrupt system is single level. Once an interrupt is detected, all further interrupt requests are latched but are not acted upon until execution of an RETR instruction re-enables the interrupt input logic. This occurs at the beginning of the second cycle of the RETR instruction. If an IBF interrupt and a timer-overflow interrupt occur simultaneously, the IBF interrupt will be recognized first and the timer-overflow interrupt will remain pending until the end of the interrupt service routine.

External Interrupts

An external interrupt can be created using the UPI-41AH, 42 timer/counter in the event counter mode. The counter is first preset to FFH and the EN TCNTI instruction is executed. A timer-overflow interrupt is generated by the first high to low transition of the TEST 1 input pin. Also, if an IBF interrupt occurs during servicing of the timer/counter interrupt, it will remain pending until the end of the service routine.

Host Interrupts And DMA

If needed, two external interrupts to the host system can be created using the EN FLAGS instruction. This instruction allocates two I/O lines on PORT 2 (P_{24} and P_{25}). P_{24} is the Output Buffer Full interrupt request line to the host system; P_{25} is the Input Buffer empty interrupt request line. These interrupt outputs reflect the internal status of the OBF flag and the IBF inverted flag. Note, these outputs may be inhibited by writing a "0" to these pins. Reenabling interrupts is done by writing a "1" to these port pins. Interrupts are typically enabled after power on since the I/O ports are set in a "1" condition. The EN FLAG's effect is only cancelled by a device RESET.

DMA handshaking controls are available from two pins on PORT 2 of the UPI-41A microcomputer. These lines (P_{26} and P_{27}) are enabled by the EN DMA instruction. P_{26} becomes DMA request ($\overline{\text{DRQ}}$) and P_{27} becomes DMA acknowledge ($\overline{\text{DACK}}$). The UPI program initiates a DMA request by writing a "1" to P_{26}. The DMA controller transfers the data into the DBBIN data register using DACK which acts as a chip select. The EN DMA instruction can only be cancelled by a chip RESET.

RESET

The $\overline{\text{RESET}}$ input provides a means for internal initialization of the processor. An automatic initialization pulse can be generated at power-on by simply connecting a 1 μfd capacitor between the $\overline{\text{RESET}}$ input and ground as shown in Figure 2-15. It has an internal pull-up resistor to charge the capacitor and a Schmitt-trigger circuit to generate a clean transition. A 2-stage sychronizer has been added to support reliable operation up to 12 MHz.

If automatic initialization is used, $\overline{\text{RESET}}$ should be held low for at least 10 milliseconds to allow the power supply to stabilize. If an external $\overline{\text{RESET}}$ signal is used, $\overline{\text{RESET}}$ may be held low for a minimum of 8 instruction cycles. Figure 2-15 illustrates a configuration using an external TTL gate to generate the $\overline{\text{RESET}}$ input. This configuration can be used to derive the $\overline{\text{RESET}}$ signal from the 8224 clock generator in an 8080 system.

The $\overline{\text{RESET}}$ input performs the following functions:

- Disables Interrupts
- Clears Program Counter to Zero
- Clears Stack Pointer
- Clears Status Register and Flags
- Clears Timer and Timer Flag
- Stops Timer
- Selects Register Bank 0
- Sets PORTS 1 and 2 to Input Mode

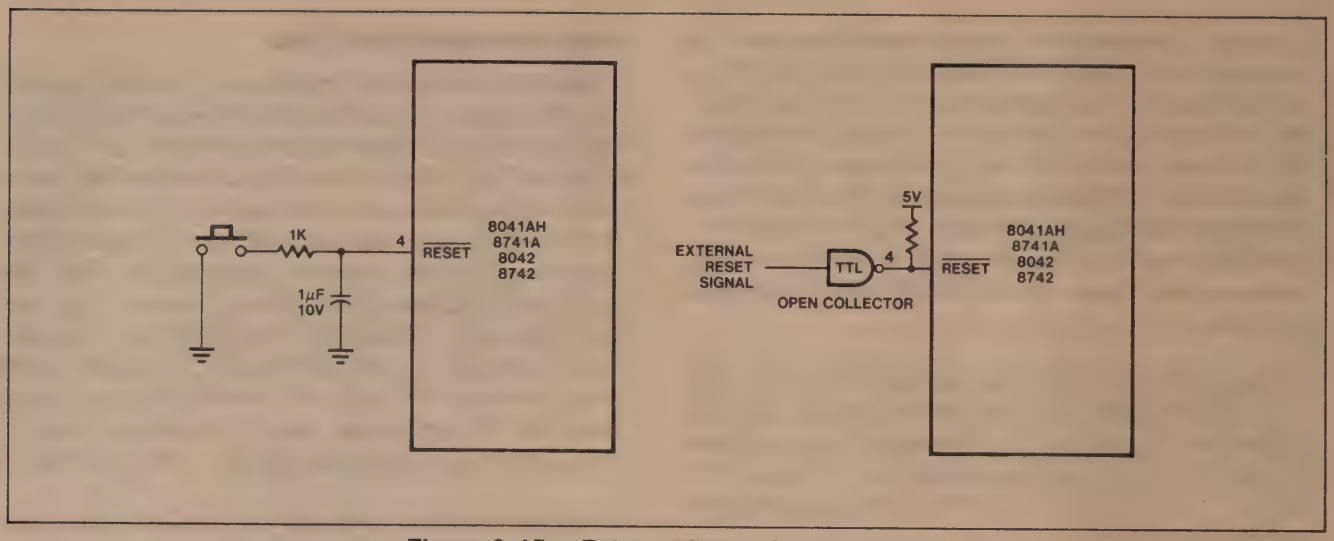


Figure 2-15. External Reset Configuration

DATA BUS BUFFER

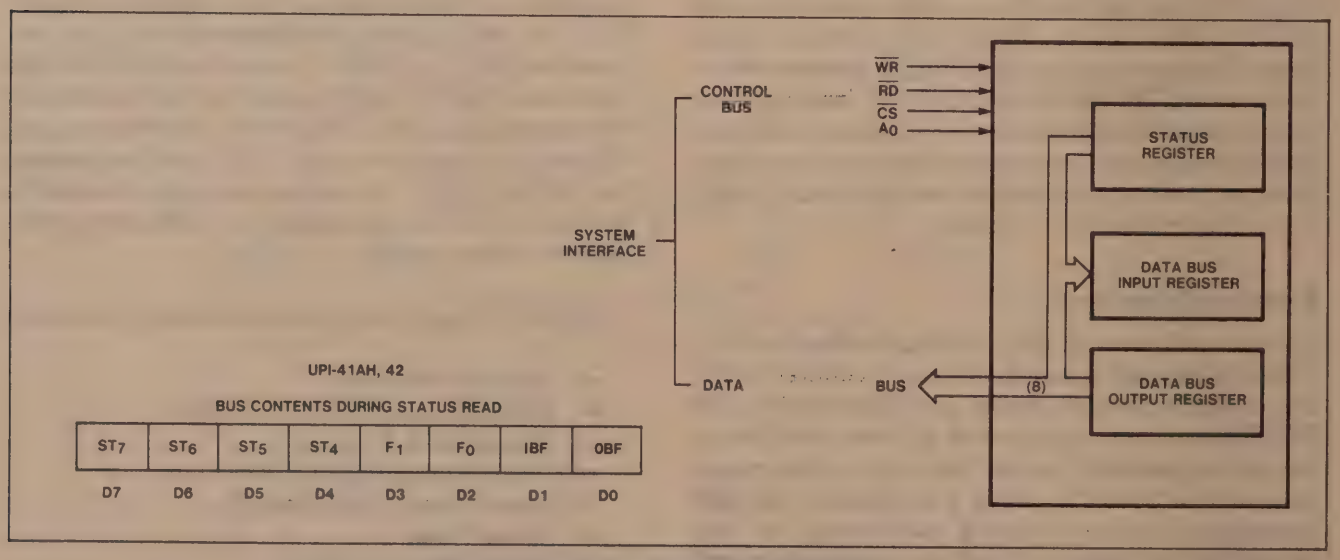
Two 8-bit data bus buffer registers, DBBIN and DBBOUT, serve as temporary buffers for commands and data flowing between it and the master processor. Externally, data is transmitted or received by the DBB registers upon execution of an INput or OUTput instruction by the master processor. Four control signals are used:

- A_0 Address input signifying control or data
- $\overline{CS}$ Chip Select
- $\overline{RD}$ Read strobe
- $\overline{WR}$ Write strobe

Transfer can be implemented with or without UPI program interference by enabling or disabling an internal UPI interrupt. Internally, data transfer between the DBB and the UPI accumulator is under software control and is completely asynchronous to the external processor timing. This allows the UPI software to handle peripheral control tasks independent of the main processor while still maintaining a data interface with the master system.

Configuration

Figure 2-16 illustrates the internal configuration of the DBB registers. Data is stored in two 8-bit buffer registers, DBBIN and DBBOUT. DBBIN and DBBOUT may be accessed by the external processor using the $\overline{WR}$ line and the $\overline{RD}$ line, respectively. The data bus is a bidirectional, three-state bus which can be connected directly to an 8-bit microprocessor system. Four control lines ($\overline{WR}$, $\overline{RD}$, $\overline{CS}$, A_0) are used by the external processor to transfer data to and from the DBBIN and DBBOUT registers.



2-16. Data Bus Buffer Configuration

An 8-bit register containing status flags is used to indicate the status of the DBB registers. The eight status flags are defined as follows:

- **OBF Output Buffer Full** This flag is automatically set when the UPI-Microcomputer loads the DBBOUT register and is cleared when the master processor reads the data register.
- **IBF Input Buffer Full** This flag is set when the master processor writes a character to the DBBIN register and is cleared when the UPI INputs the data register contents to its accumulator.
- **F_0** This is a general purpose flag which can be cleared or toggled under UPI software control. The flag is used to transfer UPI status information to the master processor.
- **F_1 Command/Data** This flag is set to the condition of the A_0 input line when the master processor writes a character to the data register. The F_1 flag can also be cleared or toggled under UPI-Microcomputer program control.
- **ST4 Through ST7** These bits are user defined status bits. They are defined by the MOV STS,A instruction.

All flags in the status register are automatically cleared by a $\overline{\text{RESET}}$ input.

SYSTEM INTERFACE

Figure 2-17 illustrates how an UPI-Microcomputer can be connected to a standard 8080-type bus system. Data lines D_0–D_7 form a three-state, bidirectional port which can be connected directly to the system data bus. The UPI bus interface has sufficient drive capability (400 μA) for small systems, however, a larger system may require buffers.

Four control signals are required to handle the data and status information transfer:

- $\overline{\text{WR}}$ I/O WRITE signal used to transfer data from the system bus to the UPI DBBIN register and set the F_1 flag in the status register.
- $\overline{\text{RD}}$ I/O READ signal used to transfer data from the DBBOUT register or status register to the system data bus.
- $\overline{\text{CS}}$ CHIP SELECT signal used to enable one 8041A out of several connected to a common bus.
- A_0 Address input used to select either the 8-bit status register or DBBOUT register during an I/O READ.
  Also, the signal is used to set the F_1 flag in the status register during an I/O WRITE.

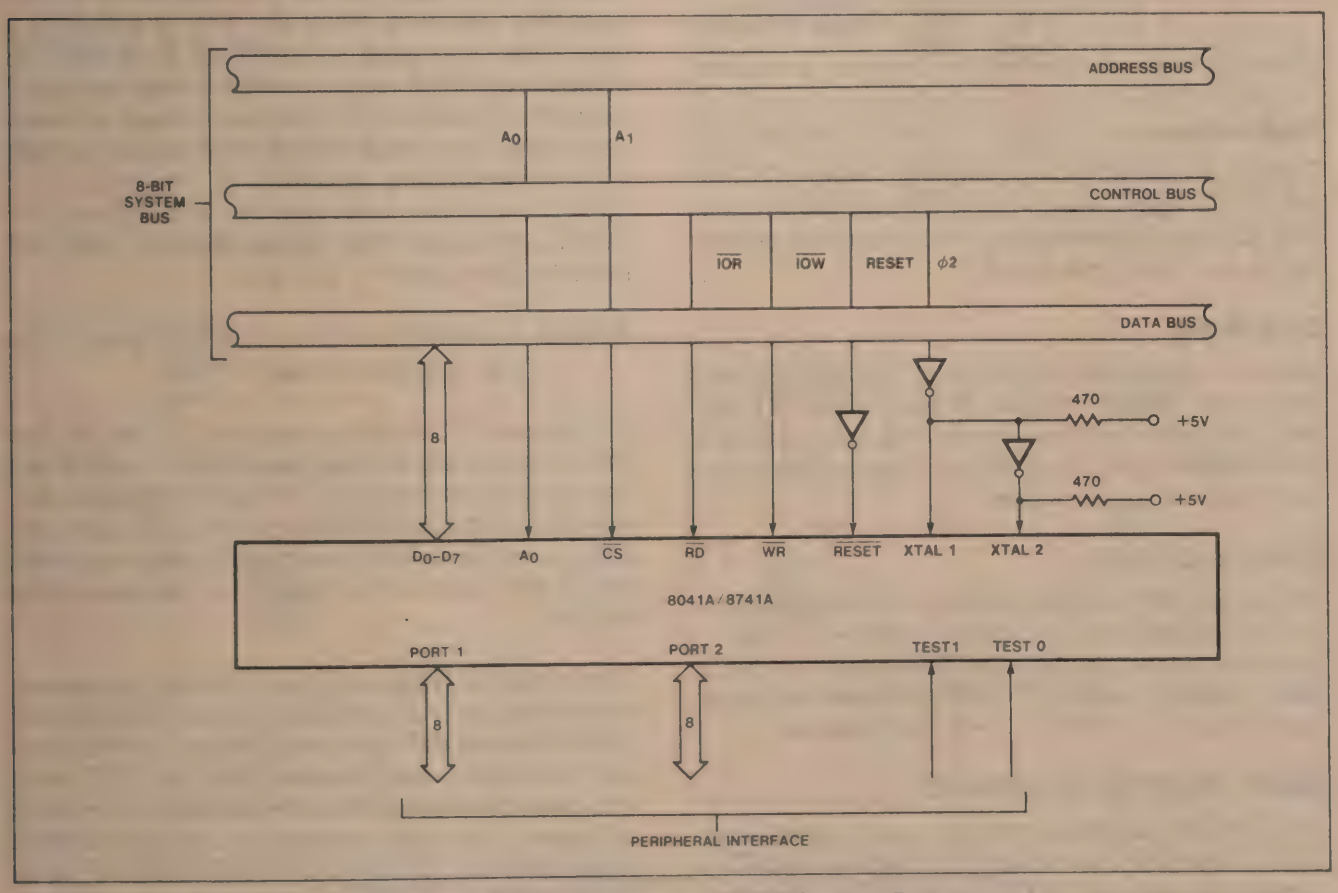


Figure 2-17. Interface to 8080 System Bus

The $\overline{WR}$ and $\overline{RD}$ signals are active low and are standard MCS-80 peripheral control signals used to synchronize data transfer between the system bus and peripheral devices.

The $\overline{CS}$ and A_0 signals are decoded from the address bus of the master system. In a system with few I/O devices a linear addressing configuration can be used where A_0 and A_1 lines are connected directly to A_0 and $\overline{CS}$ inputs (see Figure 2-17).

Data Read

Table 2-4 illustrates the relative timing of a DBBOUT Read. When $\overline{CS}$, A_0, and $\overline{RD}$ are low, the contents of the DBBOUT register is placed on the three-state Data lines D_0–D_7 and the OBF flag is cleared.

The master processor uses $\overline{CS}$, A_0, $\overline{WR}$, and $\overline{RD}$ to control data transfer between the DBBOUT register and the master system. The following operations are under master processor control:

Table 2-4. Data Transfer Controls

$\overline{CS}$	$\overline{RD}$	$\overline{WR}$	A_0	
0	0	1	0	Read DBBOUT register
0	0	1	1	Read STATUS register
0	1	0	0	Write DBBIN data register
0	1	0	1	Write DBBIN command register
1	x	x	x	Disable DBB

Status Read

Table 2-4 shows the logic sequence required for a STATUS register read. When $\overline{CS}$ and $\overline{RD}$ are low with A_0 high, the contents of the 8-bit status register appears on Data lines D_0–D_7.

Data Write

Table 2-4 shows the sequence for writing information to the DBBIN register. When $\overline{CS}$ and $\overline{WR}$ are low, the contents of the system data bus is latched into DBBIN. Also, the IBF flag is set and an interrupt is generated, if enabled.

Command Write

During any write (Table 2-4), the state of the A_0 input is latched into the status register in the F_1 (command/data) flag location. This additional bit is used to signal whether DBBIN contents are command (A_0 = 1) or data (A_0 = 0) information.

INPUT/OUTPUT INTERFACE

The UPI-41A has 16 lines for input and output functions. These I/O lines are grouped as two 8-bit TTL compatible ports: PORTS 1 and 2. The port lines can individually function as either inputs or outputs under software control. In addition, the lower 4 lines of PORT 2 can be used to interface to an 8243 I/O expander device to increase I/O capacity to 28 or more lines. The additional lines are grouped as 4-bit ports: PORTS 4, 5, 6, and 7.

PORTS 1 and 2

PORTS 1 and 2 are each 8 bits wide and have the same I/O characteristics. Data written to these ports by an OUTL Pp,A instruction is latched and remains unchanged until it is rewritten. Input data is sampled at the time the IN, A,Pp instruction is executed. Therefore, input data must be present at the PORT until read by an INput instruction. PORT 1 and 2 inputs are fully TTL compatible and outputs will drive one standard TTL load.

Circuit Configuration

The PORT 1 and 2 lines have a special output structure (shown in Figure 2-18) that allows each line to serve as an input, an output, or both, even though outputs are statically latched.

Each line has a permanent high impedance pull-up (50KΩ) which is sufficient to provide source current for a TTL high level, yet can be pulled low by a standard TTL gate drive. Whenever a "1" is written to a line, a low impedance pull-up (5K) is switched in momentarily (500 ns) to provide a fast transition from 0 to 1. When a "0" is written to the line, a low impedance pull-down (300Ω) is active to provide TTL current sinking capability.

To use a particular PORT pin as an input, a logic "1" must first be written to that pin.

NOTE: A $\overline{RESET}$ intializes all PORT pins to the high impedance logic "1" state.

An external TTL device connected to the pin has sufficient current sinking capability to pull-down the pin to the low state. An IN A,Pp instruction will sample the status of PORT pin and will input the proper logic level. With no external input connected, the IN A,Pp instruction inputs the previous output status.

This structure allows input and output information on the same pin and also allows any mix of input and output lines on the same port. However, when inputs and outputs are mixed on one PORT, a PORT write will cause the strong internal pull-ups to turn on at all inputs. If a switch or other low impedance device is connected to an input, a PORT write ("1" to an input) could cause current limits on internal lines to

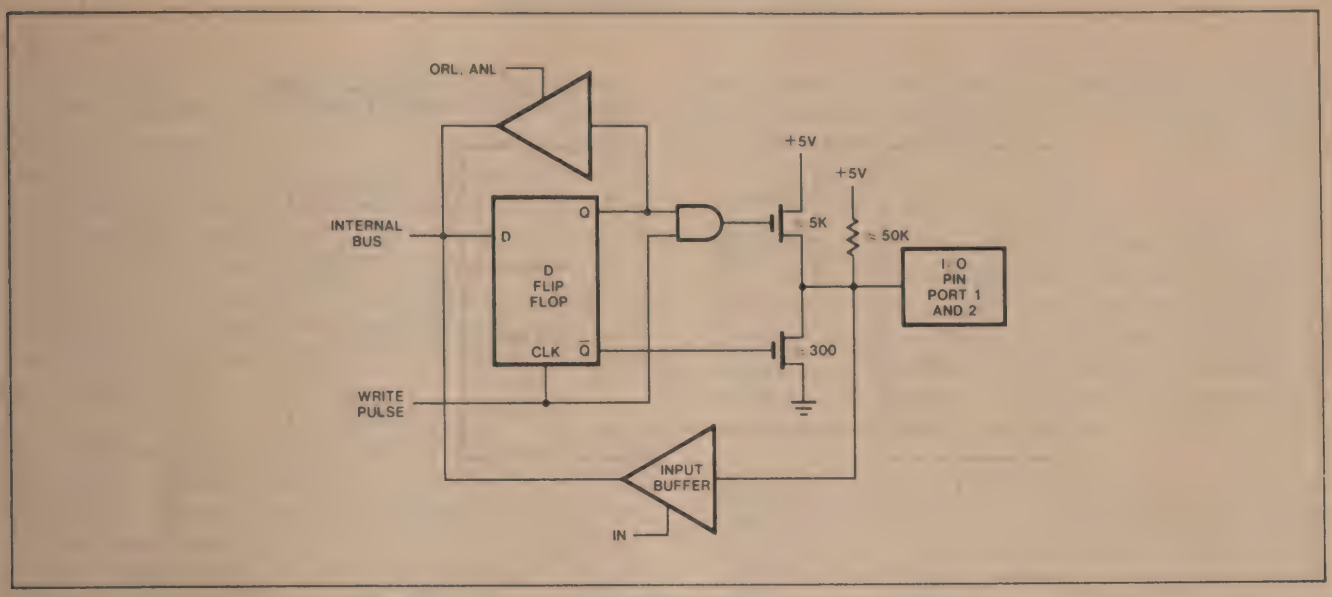


Figure 2-18. Quasi-Bidirectional Port Structure

be exceeded. Figure 2-19 illustrates the recommended connection when inputs and outputs are mixed on one PORT.

The bidirectional port structure in combination with the UPI-41AH, 42 logical AND and OR instructions provides an efficient means for handling single line inputs and outputs within an 8-bit processor.

PORTS 4, 5, 6, and 7

By using an 8243 I/O expander, 16 additional I/O lines can be connected to the UPI-41AH, 42 and directly addressed as 4-bit I/O ports using UPI-41AH, 42 instructions. This feature saves program space and design time, and improves the bit handling capability of the UPI-41AH, 42.

The lower half of PORT 2 provides an interface to the 8243 as illustrated in Figure 2-20. The PROG pin is used as a strobe to clock address and data information via the PORT 2 interface. The extra 16 I/O lines are referred to in UPI software as PORTS 4, 5, 6, and 7. Each PORT can be directly addressed and can be ANDed and ORed with an immediate data mask. Data can be moved directly to the accumulator from the expander PORTS (or vice-versa).

The 8243 I/O ports, PORTS 4, 5, 6, and 7, provide more drive capability than the UPI-41AH, 42 bidirectional ports. The 8243 output is capable of driving about 5 standard TTL loads.

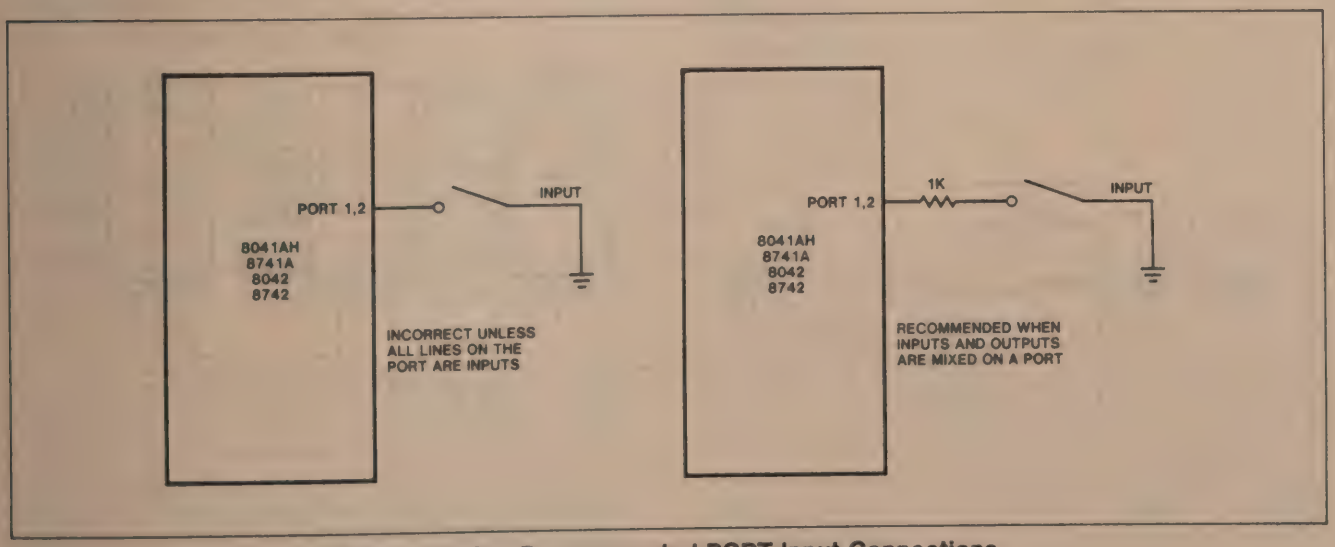


Figure 2-19. Recommended PORT Input Connections

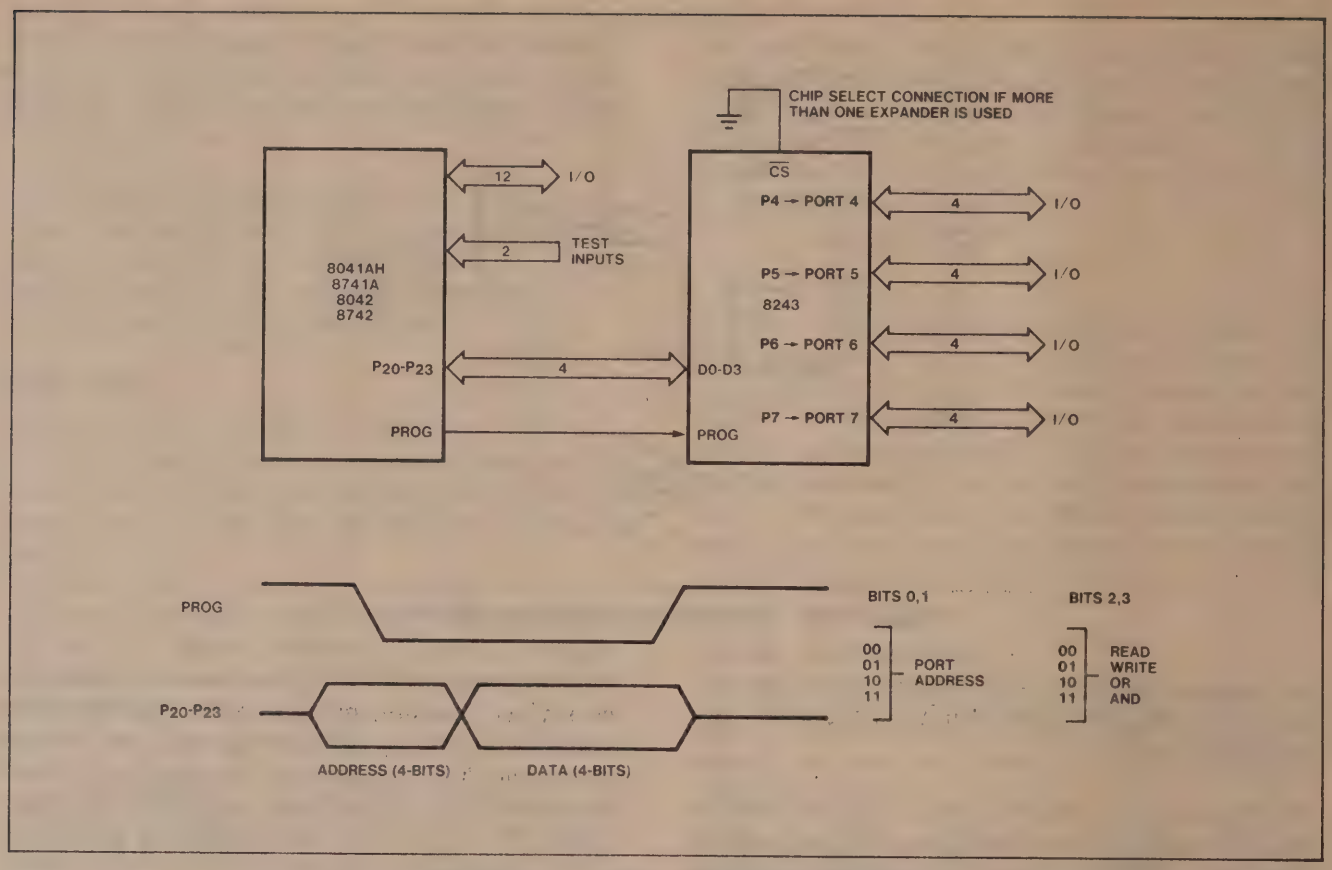


Figure 2-20. 8243 Expander Interface

Multiple 8243's can be connected to the PORT 2 interface. In normal operation, only one of the 8243's would be active at the time an Input or Output command is executed. The upper half of PORT 2 is used to provide chip select signals to the 8243's. Figure 2-21 shows how four 8243's could be connected. Software is needed to select and set the proper PORT 2 pin before an INPUT or OUTPUT command to PORTS 4–7 is executed. In general, the software overhead required is very minor compared to the added flexibility of having a large number of I/O pins available.

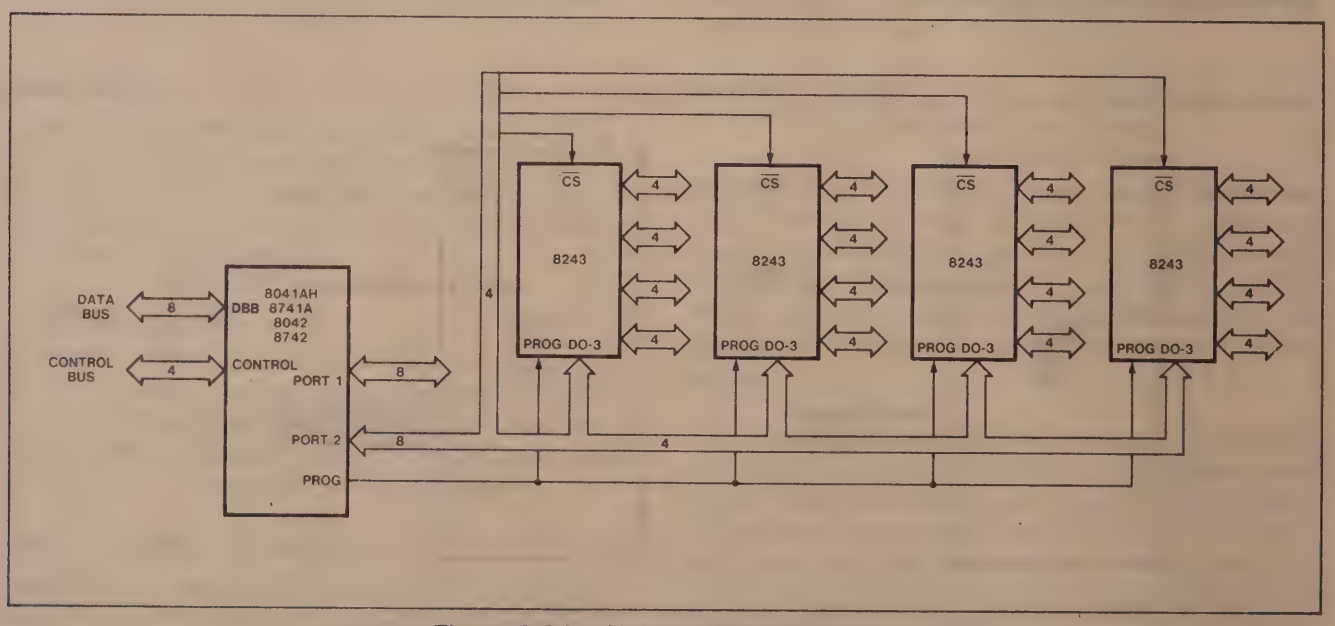


Figure 2-21. Multiple 8243 Expansion

CHAPTER 3
INSTRUCTION SET

The UPI-41AH, 42 Instruction Set is opcode-compatible with the MCS-48 set except for the elimination of external program and data memory instructions and the addition of the data bus buffer instructions. It is very straightforward and efficient in its use of program memory. All instructions are either 1 or 2 bytes in length (over 70% are only 1 byte long) and over half of the instructions execute in one machine cycle. The remainder require only two cycles and include Branch, Immediate, and I/O operations.

The UPI-41AH, 42 Instruction Set efficiently handles the single-bit operations required in control applications. Special instructions allow port bits to be set or cleared individually. Also, any accumulator bit can be directly tested via conditional branch instructions. Additional instructions are included to simplify loop counters, table look-up routines and N-way branch routines.

The UPI-41AH, 42 Microcomputer handles arithmetic operations in both binary and BCD for efficient interface to peripherals such as keyboards and displays.

The instruction set can be divided into the following groups:

- Data Moves
- Accumulator Operations
- Flags
- Register Operations
- Branch Instructions
- Control
- Timer Operations
- Subroutines
- Input/Output Instructions

Data Moves (See Instruction Summary)

The 8-bit accumulator is the control point for all data transfers within the UPI-41AH, 42. Data can be transferred between the 8 registers of each working register bank and the accumulator directly (i.e., with a source or destination register specified by 3 bits in the instruction). The remaining locations in the RAM array are addressed either by R_0 or R_1 of the active register bank. Transfers to and from RAM require one cycle.

Constants stored in Program Memory can be loaded directly into the accumulator or the eight working registers. Data can also be transferred directly between the accumulator and the on-board timer/counter, the Status Register (STS), or the Program Status Word (PSW). Transfers to the STS register alter bits 4-7 only. Transfers to the PSW alter machine status accordingly and provide a means of restoring status after an interrupt or of altering the stack pointer if necessary.

Accumulator Operations

Immediate data, data memory, or the working registers can be added (with or without carry) to the accumulator. These sources can also be ANDed, ORed, or exclusive ORed to the accumulator. Data may be moved to or from the accumulator and working registers or data memory. The two values can also be exchanged in a single operation.

The lower 4 bits of the accumulator can be exchanged with the lower 4 bits of any of the internal RAM locations. This operation, along with an instruction which swaps the upper and lower 4-bit halves of the accumulator, provides easy handling of BCD numbers and other 4-bit quantities. To facilitate BCD arithmetic a Decimal Adjust instruction is also included. This instruction is used to correct the result of the binary addition of two 2-digit BCD numbers. Performing a decimal adjust on the result in the accumulator produces the desired BCD result.

The accumulator can be incremented, decremented, cleared, or complemented and can be rotated left or right 1 bit at a time with or without carry.

A subtract operation can be easily implemented in UPI-41AH, 42 software using three single-byte, single-cycle instructions. A value can be subtracted from the accumulator by using the following instructions:

- Complement the accumulator
- Add the value to the accumulator
- Complement the accumulator

Flags

There are four user accessible flags:

- Carry
- Auxiliary Carry
- F_0
- F_1

The Carry flag indicates overflow of the accumulator, while the Auxiliary Carry flag indicates overflow between BCD digits and is used during decimal adjust operations. Both Carry and Auxiliary Carry are part of the Program Status Word (PSW) and are stored in the stack during subroutine calls. The F_0 and F_1 flags are general-purpose flags which can be cleared or complemented by UPI instructions. F_0 is accessible via the Program Status Word and is stored in the stack with the Carry flags. F_1 reflects the condition of the A_0 line, and caution must be used when setting or clearing it.

Register Operations

The working registers can be accessed via the accumulator as explained above, or they can be loaded with immediate data constants from program memory. In addition, they can be incremented or decremented directly, or they can be used as loop counters as explained in the section on branch instructions.

Additional Data Memory locations can be accessed with indirect instructions via R_0 and R_1.

Branch Instructions

The UPI-41AH, 42 Instruction Set includes 17 jump instructions. The unconditional jump instruction allows jumps anywhere in the 1K words of program memory. All other jump instructions are limited to the current page (256 words) of program memory.

Conditional jump instructions can test the following inputs and machine flags:

- TEST 0 input pin
- TEST 1 input pin
- Input Buffer Full flag
- Output Buffer Full flag
- Timer flag
- Accumulator zero
- Accumulator bit
- Carry flag
- F_0 flag
- F_1 flag

The conditions tested by these instructions are the instantaneous values at the time the conditional jump instruction is executed. For instance, the jump on accumulator zero instruction tests the accumulator itself, not an intermediate flag.

The decrement register and jump if not zero (DJNZ) instruction combines decrement and branch operations in a single instruction which is useful in implementing a loop counter. This instruction can designate any of the 8 working registers as a counter and can effect a branch to any address within the current page of execution.

A special indirect jump instruction (JMPP @A) allows the program to be vectored to any one of several different locations based on the contents of the accumulator. The contents of the accumulator point to a location in program memory which contains the jump address. As an example, this instruction could be used to vector to any one of several routines based on an ASCII character which has been loaded into the accumulator. In this way, ASCII inputs can be used to initiate various routines.

Control

The UPI-41AH, 42 Instruction Set has six instructions for control of the DMA, interrupts, and selection of working register banks.

The UPI-41AH, 42 provides two instructions for control of the external microcomputer system. IBF and OBF flags can be routed to PORT 2 allowing interrupts of the external processor. DMA handshaking signals can also be enabled using lines from PORT 2.

The IBF interrupt can be enabled and disabled using two instructions. Also, the interrupt is automatically disabled following a RESET input or during an interrupt service routine.

The working register bank switch instructions allow the programmer to immediately substitute a second 8 register bank for the one in use. This effectively provides either 16 working registers or the means for quickly saving the contents of the first 8 registers in response to an interrupt. The user has the option of switching register banks when an interrupt occurs. However, if the banks are switched, the original bank will automatically be restored upon execution of a return and restore status (RETR) instruction at the end of the interrupt service routine.

Timer

The 8-bit on-board timer/counter can be loaded or read via the accumulator while the counter is stopped or while counting.

The counter can be started as a timer with an internal clock source or as an event counter or timer with an external clock applied to the TEST 1 pin. The instruction executed determines which clock source is used. A single instruction stops the counter whether it is operating with an internal or an external clock source. In addition, two instructions allow the timer interrupt to be enabled or disabled.

Subroutines

Subroutines are entered by executing a call instruction. Calls can be made to any address in the 1K word program memory. Two separate return instructions determine whether or not status (i.e., the upper 4 bits of the PSW) is restored upon return from a subroutine.

Input/Output Instructions

Two 8-bit data bus buffer registers (DBBIN and DBBOUT) and an 8-bit status register (STS) enable the UPI-41A universal peripheral interface to communicate with the external microcomputer system. Data can be INputted from the DBBIN register to

the accumulator. Data can be OUTputted from the accumulator to the DBBOUT register.

The STS register contains four user-definable bits (ST_4–ST_7) plus four reserved status bits (IBF, OBF, F_0, and F_1). The user-definable bits are set from the accumulator.

The UPI-41AH, 42 peripheral interface has two 8-bit static I/O ports which can be loaded to and from the accumulator. Outputs are statically latched but inputs to the ports are sampled at the time an IN instruction is executed. In addition, immediate data from program memory can be ANDed and ORed directly to PORTS 1 and 2 with the result remaining on the port. This allows "masks" stored in program memory to be used to set or reset individual bits on the I/O ports. PORTS 1 and 2 are configured to allow input on a given pin by first writing a "1" to the pin.

Four additional 4-bit ports are available through the 8243 I/O expander device. The 8243 interfaces to the UPI-41AH, 42 peripheral interface via four PORT 2 lines which form an expander bus. The 8243 ports have their own AND and OR instructions like the on-board ports, as well as move instructions to transfer data in or out. The expander AND or OR instructions, however, combine the contents of the accumulator with the selected port rather than with immediate data as is done with the on-board ports.

INSTRUCTION SET DESCRIPTION

The following section provides a detailed description of each UPI instruction and illustrates how the instructions are used.

For further information about programming the UPI, consult the *8048/8041A Assembly Language Manual.*

Table 3-1. Symbols and Abbreviations Used

Symbol	Definition
A	Accumulator
C	Carry
DBBIN	Data Bus Buffer Input
DBBOUT	Data Bus Buffer Output
F_0, F_1	FLAG 0, FLAG 1 (C/D flag)
I	Interrupt
P	Mnemonic for "in-page" operation
PC	Program Counter
Pp	Port designator (p = 1, 2, or 4–7)
PSW	Program Status Word
Rr	Register designator (r = 0–7)
SP	Stack Pointer
STS	Status register
T	Timer
TF	Timer Flag
T_0, T_1	TEST 0, TEST 1
#	Immediate data prefix
@	Indirect address prefix
(( ))	Double parentheses show the effect of @, that is, @RO is shown as ((RO)).
( )	Contents of

Table 3-2. Instruction Set Summary

Mnemonic		Operation Description	Bytes	Cycles
Accumulator				
ADD	A,Rr	Add register to A	1	1
ADD	A,@Rr	Add data memory to A	1	1
ADD	A,#data	Add immediate to A	2	2
ADDC	A,Rr	Add register to A with carry	1	1
ADDC	A,@Rr	Add data memory to A with carry	1	1
ADDC	A,#data	Add immediate to A with carry	2	2
ANL	A,Rr	And register to A	1	1
ANL	A,@Rr	And data memory to A	1	1
ANL	A,#data	And immediate to A	2	2
ORL	A,Rr	Or register to A	1	1
ORL	A,@Rr	Or data memory to A	1	1
ORL	A,#data	Or immediate to A	2	2
XRL	A,Rr	Exclusive Or register to A	1	1
XRL	A,@Rr	Exclusive Or data memory to A	1	1
XRL	A,#data	Exclusive Or immediate to A	2	2
INC	A	Increment A	1	1
DEC	A	Decrement A	1	1
CLR	A	Clear A	1	1
CPL	A	Complement A	1	1
DA	A	Decimal Adjust A	1	1
SWAP	A	Swap nibbles of A	1	1
RL	A	Rotate A left	1	1
RLC	A	Rotate A left through carry	1	1
RR	A	Rotate A right	1	1
RRC	A	Rotate A right through carry	1	1

Table 3-2. Instruction Set Summary (Con't.)

Mnemonic		Operation Description	Bytes	Cycles
INPUT/OUTPUT				
IN	A,Pp	Input port to A	1	2
OUTL	Pp,A	Output A to port	1	2
ANL	Pp,#data	And immediate to port	2	2
ORL	Pp,#data	Or immediate to port	2	2
IN	A,DBB	Input DBB to A, clear IBF	1	1
OUT	DBB,A	Output A to DBB, Set OBF	1	1
MOV	STS,A	A_4–A_7 to bits 4–7 of status	1	1
MOVD	A,Pp	Input Expander port to A	1	2
MOVD	Pp,A	Output A to Expander port	1	2
ANLD	Pp,A	And A to Expander port	1	2
ORLD	Pp,A	Or A to Expander port	1	2
DATA MOVES				
MOV	A,Rr	Move register to A	1	1
MOV	A,@Rr	Move data memory to A	1	1
MOV	A,#data	Move immediate to A	2	2
MOV	Rr,A	Move A to register	1	1
MOV	@Rr,A	Move A to data memory	1	1
MOV	Rr,#data	Move immediate to register	2	2
MOV	@Rr,#data	Move immediate to data memory	2	2
MOV	A,PSW	Move PSW to A	1	1
MOV	PSW,A	Move A to PSW	1	1
XCH	A,Rr	Exchange A and registers	1	1
XCH	A,@Rr	Exchange A and data memory	1	1
XCHD	A,@Rr	Exchange digit of A and register	1	1
MOVP	A,@A	Move to A from current page	1	2
MOVP3	A,@A	Move to A from Page 3	1	2
TIMER/COUNTER				
MOV	A,T	Read Timer/Counter	1	1
MOV	T,A	Load Timer/Counter	1	1
STRT	T	Start Timer	1	1
STRT	CNT	Start Counter	1	1
STOP	TCNT	Stop Timer/Counter	1	1
EN	TCNTI	Enable Timer/Counter Interrupt	1	1
DIS	TCNTI	Disable Timer/Counter Interrupt	1	1
CONTROL				
EN	DMA	Enable DMA Handshake Lines	1	1
EN	I	Enable IBF interrupt	1	1
DIS	I	Disable IBF interrupt	1	1
EN	FLAGS	Enable Master Interrupts	1	1
SEL	RB0	Select register bank 0	1	1
SEL	RB1	Select register bank 1	1	1
NOP		No Operation	1	1
REGISTERS				
INC	Rr	Increment register	1	1
INC	@Rr	Increment data memory	1	1
DEC	Rr	Decrement register	1	1
SUBROUTINE				
CALL	addr	Jump to subroutine	2	2
RET		Return	1	2
RETR		Return and restore status	1	2
FLAGS				
CLR C		Clear Carry	1	1
CPL C		Complement Carry	1	1
CLR F0		Clear Flag 0	1	1
CPL F0		Complement Flag 0	1	1
CLR F1		Clear F_1 Flag	1	1
CPL F1		Complement F_1 Flag	1	1

Table 3-2. Instruction Set Summary (Con't.)

Mnemonic		Operation Description	Bytes	Cycles
BRANCH				
JMP	addr	Jump unconditional	2	2
JMPP	@A	Jump indirect	1	2
DJNZ	Rr,addr	Decrement register and jump on non-zero	2	2
JC	addr	Jump on Carry=1	2	2
JNC	addr	Jump on Carry=0	2	2
JZ	addr	Jump on A Zero	2	2
JNZ	addr	Jump on A not Zero	2	2
JT0	addr	Jump on T_0=1	2	2
JNT0	addr	Jump on T_0=0	2	2
JT1	addr	Jump on T_1=1	2	2
JNT1	addr	Jump on T_1=0	2	2
JF0	addr	Jump on F_0 Flag=1	2	2
JF1	addr	Jump on F_1 Flag=1	2	2
JTF	addr	Jump on Timer Flag=1	2	2
JNIBF	addr	Jump on IBF Flag=0	2	2
JOBF	addr	Jump on OBF Flag=1	2	2
JBb	addr	Jump on Accumulator Bit	2	2

ALPHABETIC LISTING

ADD A,Rr Add Register Contents to Accumulator

Opcode: | 0 1 1 0 | 1 r_2 r_1 r_0 |

The contents of register 'r' are added to the accumulator. Carry is affected.

(A) ← (A) + (Rr) r=0–7

Example:
```
ADDREG: ADD A,R6      ;ADD REG 6 CONTENTS
                      ;TO ACC
```

ADD A,@Rr Add Data Memory Contents to Accumulator

Opcode: | 0 1 1 0 | 0 0 0 r |

The contents of the standard data memory location addressed by register 'r' bits 0–5 are added to the accumulator. Carry is affected.

(A) ← (A) + ((Rr)) r=0–1

Example:
```
ADDM: MOV RO,#47      ;MOVE 47 DECIMAL TO REG 0
      ADD A,@RO       ;ADD VALUE OF LOCATION
                      ;47 TO ACC
```

ADD A,#data Add Immediate Data to Accumulator

Opcode: | 0 0 0 0 | 0 0 1 1 | • | d_7 d_6 d_5 d_4 | d_3 d_2 d_1 d_0 |

This is a 2-cycle instruction. The specified data is added to the accumulator. Carry is affected.

(A) ← (A) + data

Example:
```
ADDID: ADD A,#ADDER   ;ADD VALUE OF SYMBOL
                      ;'ADDER' TO ACC
```

ADDC A,Rr Add Carry and Register Contents to Accumulator

Opcode:

0	1	1	1	1	r_2	r_1	r_0

The content of the carry bit is added to accumulator location 0. The contents of register 'r' are then added to the accumulator. Carry is affected.

(A) ← (A) + (Rr) + (C) r=0–7

Example:

```
ADDRGC: ADDC A,R4      ;ADD CARRY AND REG 4
                       ;CONTENTS TO ACC
```

ADDC A,@Rr Add Carry and Data Memory Contents to Accumulator

Opcode:

0	1	1	1	0	0	0	r

The content of the carry bit is added to accumulator location 0. Then the contents of the standard data memory location addressed by register 'r' bits 0–5 are added to the accumulator. Carry is affected.

(A) ← (A) + ((Rr)) + (C) r=0–1

Example:

```
ADDMC: MOV R1,#40      ;MOV '40' DEC TO REG 1
       ADDC A,@R1      ;ADD CARRY AND LOCATION 40
                       ;CONTENTS TO ACC
```

ADDC A,#data Add Carry and Immediate Data to Accumulator

Opcode:

								•							
0	0	0	1	0	0	1	1	•	d_7	d_6	d_5	d_4	d_3	d_2	d_1	d_0

This is a 2-cycle instruction. The content of the carry bit is added to accumulator location 0. Then the specified data is added to the accumulator. Carry is affected.

(A) ← (A) + data + (C)

Example:

```
ADDC A,#255            ;ADD CARRY AND '225' DEC
                       ;TO ACC
```

ANL A,Rr Logical AND Accumulator With Register Mask

Opcode:

0	1	0	1	1	r_2	r_1	r_0

Data in the accumulator is logically ANDed with the mask contained in working register 'r'.

(A) ← (A) AND (Rr) r=0–7

Example:

```
ANDREG: ANL A,R3       ;'AND' ACC CONTENTS WITH MASK
                       ;MASK IN REG 3
```

ANL A,@Rr Logical AND Accumulator With Memory Mask

Opcode:

0	1	0	1	0	0	0	r

Data in the accumulator is logically ANDed with the mask contained in the data memory location referenced by register 'r', bits 0–5.

(A) ← (A) AND ((Rr)) r=0–1

Example:

```
ANDDM: MOV R0,#0FFH    ;MOVE 'FF' HEX TO REG 0
       ANL A,#0AFH     ;'AND' ACC CONTENTS WITH
                       ;MASK IN LOCATION 63
```

ANL A,#data Logical AND Accumulator With Immediate Mask

Opcode: | 0 1 0 1 | 0 0 1 1 | • | d_7 d_6 d_5 d_4 | d_3 d_2 d_1 d_0 |

This is a 2-cycle instruction. Data in the accumulator is logically ANDed with an immediately-specified mask.
(A) ← (A) AND data

Example:

```
ANDID: ANL A,#0AFH      ;'AND' ACC CONTENTS
                        :WITH MASK 10101111
       ANL A,#3+X/Y     ;'AND' ACC CONTENTS
                        ;WITH VALUE OF EXP
                        ;'3+X/Y'
```

ANL Pp,#data Logical AND Port 1–2 With Immediate Mask

Opcode: | 1 0 0 1 | 1 0 p_1 p_0 | • | d_7 d_6 d_5 d_4 | d_3 d_2 d_1 d_0 |

This is a 2-cycle instruction. Data on port 'p' is logically ANDed with an immediately-specified mask.
(Pp) ← (Pp) AND data p=1–2

Note: Bits 0-1 of the opcode are used to represent PORT 1 and PORT 2. If you are coding in binary rather than assembly language, the mapping is as follows:

Bits	p1	p0	Port
	0	0	X
	0	1	1
	1	0	2
	1	1	X

Example:

```
ANDP2: ANL P2,#OF0H     ;'AND' PORT 2 CONTENTS
                        ;WITH MASK 'F0' HEX
                        ;(CLEAR P20–23)
```

ANLD Pp,A Logical AND Port 4–7 With Accumulator Mask

Opcode: | 1 0 0 1 | 1 1 p_1 p_0 |

This is a 2-cycle instruction. Data on port 'p' on the 8243 expander is logically ANDed with the digit mask contained in accumulator bits 0–3.
(Pp) ← (Pp) AND (A0–3) p=4–7

Note: The mapping of Port 'p' to opcode bits p_1,p_0 is as follows:

p1	p0	Port
0	0	4
0	1	5
1	0	6
1	1	7

Example:

```
ANDP4: ANLD P4,A        ;'AND' PORT 4 CONTENTS
                        ;WITH ACC BITS 0–3
```

CALL address Subroutine Call

Opcode:

0	a_9	a_8	1	0	1	0	0	•	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0

This is a 2-cycle instruction. The program counter and PSW bits 4–7 are saved in the stack. The stack pointer (PSW bits 0–2) is updated. Program control is then passed to the location specified by 'address'.

Execution continues at the instruction following the CALL upon return from the subroutine.

$((SP)) \leftarrow (PC), (PSW_{4-7})$

$(SP) \leftarrow (SP) + 1$

$(PC_{8-9}) \leftarrow (addr_{8-9})$

$(PC_{0-7}) \leftarrow (addr_{0-7})$

Example: Add three groups of two numbers. Put subtotals in locations 50, 51 and total in location 52.

```
          MOV R0,#50        ;MOVE '50' DEC TO ADDRESS
                            ;REG 0
BEGADD: MOV A,R1            ;MOVE CONTENTS OF REG 1
                            ;TO ACC
          ADD A,R2          ;ADD REG 2 TO ACC
          CALL SUBTOT       ;CALL SUBROUTINE 'SUBTOT'
          ADD A,R3          ;ADD REG 3 TO ACC
          ADD A,R4          ;ADD REG 4 TO ACC
          CALL SUBTOT       ;CALL SUBROUTINE 'SUBTOT'
          ADD A,R5          ;ADD REG 5 TO ACC
          ADD A,R6          ;ADD REG 6 TO ACC
          CALL SUBTOT       ;CALL SUBROUTINE 'SUBTOT'
                .
                .
                .
SUBTOT: MOV @R0,A           ;MOVE CONTENTS OF ACC TO
                            ;LOCATION ADDRESSED BY
                            ;REG 0
          INC R0            ;INCREMENT REG 0
          RET               ;RETURN TO MAIN PROGRAM
```

CLR A Clear Accumulator

Opcode:

0	0	1	0	0	1	1	1

The contents of the accumulator are cleared to zero.

$(A) \leftarrow 00H$

CLR C Clear Carry Bit

Opcode:

1	0	0	1	0	1	1	1

During normal program execution, the carry bit can be set to one by the ADD, ADDC, RLC, CPLC, RRC, and DAA instructions. This instruction resets the carry bit to zero.

$(C) \leftarrow 0$

CLR F1 Clear Flag 1

Opcode:

1	0	1	0	0	1	0	1

The F_1 flag is cleared to zero.

$(F_1) \leftarrow 0$

CLR F0 Clear Flag 0

Opcode:

1	0	0	0	0	1	0	1

Flag 0 is cleared to zero.
$(F_0) \leftarrow 0$

CPL A Complement Accumulator

Opcode:

0	0	1	1	0	1	1	1

The contents of the accumulator are complemented. This is strictly a one's complement. Each one is changed to zero and vice-versa.
(A) ← NOT (A)

Example: Assume accumulator contains 01101010.

```
CPLA: CPL A          ;ACC CONTENTS ARE COMPLE-
                     ;MENTED TO 10010101
```

CPL C Complement Carry Bit

Opcode:

1	0	1	0	0	1	1	1

The setting of the carry bit is complemented; one is changed to zero, and zero is changed to one.
(C) ← NOT (C)

Example: Set C to one; current setting is unknown.

```
CT01: CLR C          ;C IS CLEARED TO ZERO
      CPL C          ;C IS SET TO ONE
```

CPL F0 Complement Flag 0

Opcode:

1	0	0	1	0	1	0	1

The setting of Flag 0 is complemented; one is changed to zero, and zero is changed to one.
$F_0 \leftarrow$ NOT (F0)

CPL F1 Complement Flag 1

Opcode:

1	0	1	1	0	1	0	1

The setting of the F_1 Flag is complemented; one is changed to zero, and zero is changed to one.
$(F_1) \leftarrow$ NOT (F_1)

DA A Decimal Adjust Accumulator

Opcode:

0	1	0	1	0	1	1	1

The 8-bit accumulator value is adjusted to form two 4-bit Binary Coded Decimal (BCD) digits following the binary addition of BCD numbers. The carry bit C is affected. If the contents of bits 0–3 are greater than nine, or if AC is one, the accumulator is incremented by six.

The four high-order bits are then checked. If bits 4–7 exceed nine, or if C is one, these bits are increased by six. If an overflow occurs, C is set to one; otherwise, it is cleared to zero.

Example: Assume accumulator contains 9AH.

```
DA   A    ;ACC ADJUSTED TO 01H with C set

C    AC   ACC
0    0    9AH    INITIAL CONTENTS
          06H    ADD SIX TO LOW DIGIT
0    0    A1H
          60H    ADD SIX TO HIGH DIGIT
1    0    01H    RESULT
```

DEC A Decrement Accumulator

Opcode:

0	0	0	0	0	1	1	1

The contents of the accumulator are decremented by one.
(A) ← (A) − 1

Example: Decrement contents of data memory location 63.

```
MOV R0,#3FH     ;MOVE '3F' HEX TO REG 0
MOV A,@R0       ;MOVE CONTENTS OF LOCATION 63
                ;TO ACC
DEC A           ;DECREMENT ACC
MOV @R0,A       ;MOVE CONTENTS OF ACC TO
                ;LOCATION 63
```

DEC Rr Decrement Register

Opcode:

1	1	0	0	1	r_2	r_1	r_0

The contents of working register 'r' are decremented by one.
(Rr) ← (Rr) − 1 r=0–7

Example:

```
DECR1: DEC R1     ;DECREMENT ADDRESS REG 1
```

DIS I Disable IBF Interrupt

Opcode:

0	0	0	1	0	1	0	1

The input Buffer Full interrupt is disabled. The interrupt sequence is not initiated by $\overline{WR}$ and $\overline{CS}$, however, an IBF interrupt request is latched and remains pending until an EN I (enable IBF interrupt) instruction is executed.

Note: The IBF flag is set and cleared independent of the IBF interrupt request so that handshaking protocol can continue normally.

DIS TCNTI Disable Timer/Counter Interrupt

Opcode: | 0 0 1 1 | 0 1 0 1 |

The timer/counter interrupt is disabled. Any pending timer interrupt request is cleared. The interrupt sequence is not initiated by an overflow, but the timer flag is set and time accumulation continues.

DJNZ Rr, address Decrement Register and Test

Opcode: | 1 1 1 0 | 1 r_2 r_1 r_0 | • | a_7 a_6 a_5 a_4 | a_3 a_2 a_1 a_0 |

This is a 2-cycle instruction. Register 'r' is decremented and tested for zero. If the register contains all zeros, program control falls through to the next instruction. If the register contents are not zero, control jumps to the specified address within the current page.

(Rr) ← (Rr) − 1

If R ≠ 0, then;

(PC_{0-7}) ← addr

Note: A 10-bit address specification does not cause an error if the DJNZ instruction and the jump target are on the same page. If the DJNZ instruction begins in location 255 of a page, it will jump to a target address on the following page. Otherwise, it is limited to a jump within the current page.

Example: Increment values in data memory locations 50–54.

```
        MOV R0,#50          ;MOVE '50' DEC TO ADDRESS
                            ;REG 0
        MOV R3,#05          ;MOVE '5' DEC TO COUNTER
                            ;REG 3
INCRT:  INC @R0             ;INCREMENT CONTENTS OF
                            ;LOCATION ADDRESSED BY
                            ;REG 0
        INC R0              ;INCREMENT ADDRESS IN REG 0
        DJNZ R3,INCRT       ;DECREMENT REG 3——JUMP TO
                            ;'INCRT' IF REG 3 NONZERO
        NEXT——              ;'NEXT' ROUTINE EXECUTED
                            ;IF R3 IS ZERO
```

EN DMA Enable DMA Handshake Lines

Opcode: | 1 1 1 0 | 0 1 0 1 |

DMA handshaking is enabled using P_{26} as DMA request (DRQ) and P_{27} as DMA acknowledge ($\overline{DACK}$). The $\overline{DACK}$ line forces $\overline{CS}$ and A_0 low internally and clears DRQ.

EN FLAGS Enable Master Interrupts

Opcode: | 1 1 1 1 | 0 1 0 1 |

The Output Buffer Full (OBF) and the Input Buffer Full (IBF) flags (IBF is inverted) are routed to P_{24} and P_{25}. For proper operation, a "1" should be written to P_{25} and P_{24} before the EN FLAGS instruction. A "0" written to P_{24} or P_{25} disables the pin.

EN I Enable IBF Interrupt

Opcode:

0	0	0	0	0	1	0	1

The Input Buffer Full interrupt is enabled. A low signal on $\overline{WR}$ and $\overline{CS}$ initiates the interrupt sequence.

EN TCNTI Enable Timer/Counter Interrupt

Opcode:

0	0	1	0	0	1	0	1

The timer/counter interrupt is enabled. An overflow of this register initiates the interrupt sequence.

IN A,DBB Input Data Bus Buffer Contents to Accumulator

Opcode:

0	0	1	0	0	0	1	0

Data in the DBBIN register is transferred to the accumulator and the Input Buffer Full (IBF) flag is set to zero.
(A) ← (DBB)
(IBF) ← 0

Example:

```
INDBB: IN A,DBB        ;INPUT DBBIN CONTENTS TO
                       ;ACCUMULATOR
```

IN A,Pp Input Port 1–2 Data to Accumulator

Opcode:

0	0	0	0	1	0	p_1	p_0

This is a 2-cycle instruction. Data present on port 'p' is transferred (read) to the accumulator.
(A) ← (Pp) p=1–2 (see ANL instruction)

Example:

```
INP12: IN A,P1         ;INPUT PORT 1 CONTENTS
                       ;TO ACC
       MOV R6,A        ;MOVE ACC CONTENTS TO
                       ;REG 6
       IN A,P2         ;INPUT PORT 2 CONTENTS
                       ;TO ACC
       MOV R7,A        ;MOVE ACC CONTENTS TO REG 7
```

INC A Increment Accumulator

Opcode:

0	0	0	1	0	1	1	1

The contents of the accumulator are incremented by one.
(A) ← (A) + 1

Example: Increment contents of location 10 in data memory.

```
INCA: MOV R0,#10       ;MOV '10' DEC TO ADDRESS
                       ;REG 0
      MOV A,@R0        ;MOVE CONTENTS OF LOCATION
                       ;10 TO ACC
      INC A            ;INCREMENT ACC
      MOV @R0,A        ;MOVE ACC CONTENTS TO
                       ;LOCATION 10
```

INC Rr Increment Register

Opcode: | 0 0 0 1 | 1 r_2 r_1 r_0 |

The contents of working register 'r' are incremented by one.

(Rr) ← (Rr) + 1 r=0–7

Example: INCR0: INC R0 ;INCREMENT ADDRESS REG 0

INC @Rr Increment Data Memory Location

Opcode: | 0 0 0 1 | 0 0 0 r |

The contents of the resident data memory location addressed by register 'r' bits 0–5 are incremented by one.

((Rr)) ← ((Rr)) + 1 r=0–1

Example: INCDM: MOV R1,#OFFH ;MOVE ONES TO REG 1
INC @R1 ;INCREMENT LOCATION 63

JBb address Jump If Accumulator Bit is Set

Opcode: | b_2 b_1 b_0 1 | 0 0 1 0 | • | a_7 a_6 a_5 a_4 | a_3 a_2 a_1 a_0 |

This is a 2-cycle instruction. Control passes to the specified address if accumulator bit 'b' is set to one.

(PC_{0-7}) ← addr if b=1
(PC) ← (PC) + 2 if b=0

Example: JB4IS1: JB4 NEXT ;JUMP TO 'NEXT' ROUTINE
;IF ACC BIT 4=1

JC address Jump If Carry Is Set

Opcode: | 1 1 1 1 | 0 1 1 0 | • | a_7 a_6 a_5 a_4 | a_3 a_2 a_1 a_0 |

This is a 2-cycle instruction. Control passes to the specified address if the carry bit is set to one.

(PC_{0-7}) ← addr if C=1
(PC) ← (PC) + 2 if C=0

Example: JC1: JC OVERFLOW ;JUMP TO 'OVFLOW' ROUTINE
;IF C=1

JF0 address Jump If Flag 0 Is Set

Opcode: | 1 0 1 1 | 0 1 1 0 | • | a_7 a_6 a_5 a_4 | a_3 a_2 a_1 a_0 |

This is a 2-cycle instruction. Control passes to the specified address if flag 0 is set to one.

(PC_{0-7}) ← addr if F_0=1

Example: JF0IS1: JF0 TOTAL ;JUMP TO 'TOTAL' ROUTINE
;IF F_0=1

JF1 address Jump If C/D Flag (F1) Is Set

Opcode: | 0 1 1 1 | 0 1 1 0 | • | a_7 a_6 a_5 a_4 | a_3 a_2 a_1 a_0 |

This is a 2-cycle instruction. Control passes to the specified address if the C/D flag (F_1) is set to one.

(PC_{0-7}) ← addr if $F_1=1$

Example:
```
JF 1IS1: JF1 FILBUF      ;JUMP TO 'FILBUF'
                         ;ROUTINE IF F1=1
```

JMP address Direct Jump Within 1K Block

Opcode: | a_{10} a_9 a_8 0 | 0 1 0 0 | • | a_7 a_6 a_5 a_4 | a_3 a_2 a_1 a_0 |

This is a 2-cycle instruction. Bits 0–9 of the program counter are replaced with the directly-specified address.

(PC_{8-9}) ← addr 8–9

(PC_{0-7}) ← addr 0–7

Example:
```
JMP SUBTOT       ;JUMP TO SUBROUTINE 'SUBTOT'
JMP $-6          ;JUMP TO INSTRUCTION SIX LOCATIONS
                 ;BEFORE CURRENT LOCATION
JMP 2FH          ;JUMP TO ADDRESS '2F' HEX
```

JMPP @A Indirect Jump Within Page

Opcode: | 1 0 1 1 | 0 0 1 1 |

This is a 2-cycle instruction. The contents of the program memory location pointed to by the accumulator are substituted for the 'page' portion of the program counter (PC 0–7).

(PC_{0-7}) ← ((A))

Example:
```
Assume accumulator contains OFH
JMPPAG: JMPP @A          ;JMP TO ADDRESS STORED IN
                         ;LOCATION 15 IN CURRENT PAGE
```

JNC address Jump If Carry Is Not Set

Opcode: | 1 1 1 0 | 0 1 1 0 | • | a_7 a_6 a_5 a_4 | a_3 a_2 a_1 a_0 |

This is a 2-cycle instruction. Control passes to the specified address if the carry bit is not set, that is, equals zero.

(PC_{0-7}) ← addr if C=0

Example:
```
JC0: JNC NOVFLO          ;JUMP TO 'NOVFLO' ROUTINE
                         ;IF C=0
```

JNIBF address Jump If Input Buffer Full Flag Is Low

Opcode: | 1 1 0 1 | 0 1 1 0 | • | a_7 a_6 a_5 a_4 | a_3 a_2 a_1 a_0 |

This is a 2-cycle instruction. Control passes to the specified address if the Input Buffer Full flag is low (IBF=0).

(PC_{0-7}) ← addr if IBF=0

Example:
```
LOC 3:JNIBF LOC 3        ;JUMP TO SELF IF IBF=0
                         ;OTHERWISE CONTINUE
```

JNTO address Jump If TEST 0 Is Low

Opcode: | 0 0 1 0 | 0 1 1 0 | • | a_7 a_6 a_5 a_4 | a_3 a_2 a_1 a_0 |

This is a 2-cycle instruction. Control passes to the specified address, if the TEST 0 signal is low. Pin is sampled during SYNC.

$(PC_{0-7}) \leftarrow$ addr if $T_0=0$

Example: JT0LOW: JNT0 60 ;JUMP TO LOCATION 60 DEC
;IF $T_0=0$

JNT1 address Jump If TEST 1 Is Low

Opcode: | 0 1 0 0 | 0 1 1 0 | • | a_7 a_6 a_5 a_4 | a_3 a_2 a_1 a_0 |

This is a 2-cycle instruction. Control passes to the specified address if the TEST 1 signal is low. Pin is sampled during SYNC.

$(PC_{0-7}) \leftarrow$ addr if $T_1=0$

Example: JT1LOW: JNT1 OBBH ;JUMP TO LOCATION 'BB' HEX
;IF $T_1=0$

JNZ address Jump If Accumulator Is Not Zero

Opcode: | 1 0 0 1 | 0 1 1 0 | • | a_7 a_6 a_5 a_4 | a_3 a_2 a_1 a_0 |

This is a 2-cycle instruction. Control passes to the specified address if the accumulator contents are nonzero at the time this instruction is executed.

$(PC_{0-7}) \leftarrow$ addr if $A \neq 0$

Example: JACCNO: JNZ OABH ;JUMP TO LOCATION 'AB' HEX
;IF ACC VALUE IS NONZERO

JOBF Address Jump If Output Buffer Full Flag Is Set

Opcode: | 1 0 0 0 | 0 1 1 0 | • | a_7 a_6 a_5 a_4 | a_3 a_2 a_1 a_0 |

This is a 2-cycle instruction. Control passes to the specified address if the Output Buffer Full (OBF) flag is set (=1) at the time this instruction is executed.

$(PC_{0-7}) \leftarrow$ addr if OBF=1

Example: JOBFHI: JOBF OAAH ;JUMP TO LOCATION 'AA' HEX
;IF OBF=1

JTF address Jump If Timer Flag Is Set

Opcode: | 0 0 0 1 | 0 1 1 0 | • | a_7 a_6 a_5 a_4 | a_3 a_2 a_1 a_0 |

This is a 2-cycle instruction. Control passes to the specified address if the timer flag is set to one, that is, the timer/counter register overflows to zero. The timer flag is cleared upon execution of this instruction. (This overflow initiates an interrupt service sequence if the timer-overflow interrupt is enabled.)

$(PC_{0-7}) \leftarrow$ addr if TF=1

Example: JTF1: JTF TIMER ;JUMP TO 'TIMER' ROUTINE
;IF TF=1

JTO address Jump If TEST 0 Is High

Opcode:	0 0 1 1	0 1 1 0	•	a_7 a_6 a_5 a_4	a_3 a_2 a_1 a_0

This is a 2-cycle instruction. Control passes to the specified address if the TEST 0 signal is high (=1). Pin is sampled during SYNC.

$(PC_{0-7}) \leftarrow$ addr if $T_0=1$

Example: JT0HI: JT0 53 ;JUMP TO LOCATION 53 DEC
;IF $T_0=1$

JT1 address Jump If TEST 1 Is High

Opcode:	0 1 0 1	0 1 1 0	•	a_7 a_6 a_5 a_4	a_3 a_2 a_1 a_0

This is a 2-cycle instruction. Control passes to the specified address if the TEST 1 signal is high (=1). Pin is sampled during SYNC.

$(PC_{0-7}) \leftarrow$ addr if $T_1=1$

Example: JT1HI: JT1 COUNT ;JUMP TO 'COUNT' ROUTINE
;IF $T_1=1$

JZ address Jump If Accumulator Is Zero

Opcode:	1 1 0 0	0 1 1 0	•	a_7 a_6 a_5 a_4	a_3 a_2 a_1 a_0

This is a 2-cycle instruction. Control passes to the specified address if the accumulator contains all zeros at the time this instruction is executed.

$(PC_{0-7}) \leftarrow$ addr if A=0

Example: JACCO: JZ OA3H ;JUMP TO LOCATION 'A3' HEX
;IF ACC VALUE IS ZERO

MOV A,#data Move Immediate Data to Accumulator

Opcode:	0 0 1 0	0 0 1 1	•	d_7 d_6 d_5 d_4	d_3 d_2 d_1 d_0

This is a 2-cycle instruction. The 8-bit value specified by 'data' is loaded in the accumulator.

(A) ← data

Example: MOV A,#OA3H ;MOV 'A3' HEX TO ACC

MOV A,PSW Move PSW Contents to Accumulator

Opcode:	1 1 0 0	0 1 1 1

The contents of the program status word are moved to the accumulator.

(A) ← (PSW)

Example: Jump to 'RB1SET' routine if bank switch, PSW bit 4, is set.

```
BSCHK: MOV A,PSW     ;MOV PSW CONTENTS TO ACC
       JB4 RB1 SET   ;JUMP TO 'RB1SET' IF ACC
                     ;BIT 4=1
```

MOV A, Rr Move Register Contents to Accumulator

Opcode:

1	1	1	1	1	r_2	r_1	r_0

Eight bits of data are moved from working register 'r' into the accumulator.

(A) ← (Rr) r=0–7

Example:

```
MAR: MOV A,R3        ;MOVE CONTENTS OF REG 3
                     ;TO ACC
```

MOV A,@Rr Move Data Memory Contents to Accumulator

Opcode:

1	1	1	1	0	0	0	r

The contents of the data memory location addressed by bits 0–5 of register 'r' are moved to the accumulator. Register 'r' contents are unaffected.

(A) ← ((Rr)) r=0–1

Example: Assume R1 contains 00110110.

```
MADM: MOV A,@R1      ;MOVE CONTENTS OF DATA MEM
                     ;LOCATION 54 TO ACC
```

MOV A,T Move Timer/Counter Contents to Accumulator

Opcode:

0	1	0	0	0	0	1	0

The contents of the timer/event-counter register are moved to the accumulator. The timer/event-counter is not stopped.

(A) ← (T)

Example: Jump to "EXIT" routine when timer reaches '64', that is, when bit 6 is set—assuming initialization to zero.

```
TIMCHK: MOV A,T      ;MOVE TIMER CONTENTS TO
                     ;ACC
        JB6 EXIT     ;JUMP TO 'EXIT' IF ACC BIT
                     ;6=1
```

MOV PSW,A Move Accumulator Contents to PSW

Opcode:

1	1	0	1	0	1	1	1

The contents of the accumulator are moved into the program status word. All condition bits and the stack pointer are affected by this move.

(PSW) ← (A)

Example: Move up stack pointer by two memory locations, that is, increment the pointer by one.

```
INCPTR: MOV A,PSW    ;MOVE PSW CONTENTS TO ACC
        INC A        ;INCREMENT ACC BY ONE
        MOV PSW,A    ;MOVE ACC CONTENTS TO PSW
```

MOV Rr,A Move Accumulator Contents to Register

Opcode: | 1 0 1 0 | 1 r_2 r_1 r_0 |

The contents of the accumulator are moved to register 'r'.
(Rr) ← (A) r=0–7

Example:
```
MRA MOV R0,A        ;MOVE CONTENTS OF ACC TO
                    ;REG 0
```

MOV Rr,#data Move Immediate Data to Register

Opcode: | 1 0 1 1 | 1 r_2 r_1 r_0 | • | d_7 d_6 d_5 d_4 | d_3 d_2 d_1 d_0 |

This a 2-cycle instruction. The 8-bit value specified by 'data' is moved to register 'r'.
(Rr) ← data r=0–7

Example:
```
MIR4: MOV R4,#HEXTEN    ;THE VALUE OF THE SYMBOL
                        ;'HEXTEN' IS MOVED INTO
                        ;REG 4
MIR5: MOV R5;#PI*(R*R)  ;THE VALUE OF THE
                        ;EXPRESSION 'PI*(R*R)'
                        ;IS MOVED INTO REG 5
MIR6: MOV R6,#OADH      ;'AD' HEX IS MOVED INTO
                        ;REG 6
```

MOV @Rr,A Move Accumulator Contents to Data Memory

Opcode: | 1 0 1 0 | 0 0 0 r |

The contents of the accumulator are moved to the data memory location whose address is specified by bits 0–5 of register 'r'. Register 'r' contents are unaffected.
((Rr)) ← (A) r=0–1

Example:
Assume R0 contains 11000111.
```
MDMA: MOV @R,A      ;MOVE CONTENTS OF ACC TO
                    ;LOCATION 7 (REG)
```

MOV @Rr,#data Move Immediate Data to Data Memory

Opcode: | 1 0 1 1 | 0 0 0 r | • | d_7 d_6 d_5 d_4 | d_3 d_2 d_1 d_0 |

This is a 2-cycle instruction. The 8-bit value specified by 'data' is moved to the standard data memory location addressed by register 'r', bit 0–5.
((Rr)) ← data r=0–1

Example:
Move the hexadecimal value AC3F to locations 62–63.
```
MIDM: MOV R0,#62        ;MOVE '62' DEC TO ADDR REG0
      MOV @R0,#OACH     ;MOVE 'AC' HEX TO LOCATION 62
      INC R0            ;INCREMENT REG 0 TO '63'
      MOV @R0,#3FH      ;MOVE '3F' HEX TO LOCATION 63
```

MOV STS,A Move Accumulator Contents to STS Register

Opcode:

1	0	0	1	0	0	0	0

The contents of the accumulator are moved into the status register. Only bits 4–7 are affected.
$(STS_{4-7}) \leftarrow (A_{4-7})$

Example: Set ST_4–ST_7 to "1".

```
MSTS: MOV A,#0F0H        ;SET ACC
      MOV STS,A          ;MOVE TO STS
```

MOV T,A Move Accumulator Contents to Timer/Counter

Opcode:

0	1	1	0	0	0	1	0

The contents of the accumulator are moved to the timer/event-counter register.
$(T) \leftarrow (A)$

Example: Initialize and start event counter.

```
INITEC: CLR A            ;CLEAR ACC TO ZEROS
        MOV T,A          ;MOVE ZEROS TO EVENT COUNTER
        STRT CNT         ;START COUNTER
```

MOVD A,Pp Move Port 4–7 Data to Accumulator

Opcode:

0	0	0	0	1	1	p_1	p_0

This is a 2-cycle instruction. Data on 8243 port 'p' is moved (read) to accumulator bits 0–3. Accumulator bits 4–7 are zeroed.
$(A_{0-3}) \leftarrow Pp$ p=4–7
$(A_{4-7}) \leftarrow 0$

Note: Bits 0–1 of the opcode are used to represent PORTS 4–7. If you are coding in binary rather than assembly language, the mapping is as follows:

Bits p_1	p_0	Port
0	0	4
0	1	5
1	0	6
1	1	7

Example:

```
INPPT5: MOVD A,P5        ;MOVE PORT 5 DATA TO ACC
                         ;BITS 0-3, ZERO ACC BITS 4-7
```

MOVD Pp,A Move Accumulator Data to Port 4, 5, 6 and 7

Opcode:

0	0	1	1	1	1	p_1	p_0

This is a 2-cycle instruction. Data in accumulator bits 0–3 is moved (written) to 8243 port 'p'. Accumulator bits 4–7 are unaffected. (See NOTE above regarding port mapping.)
$(Pp) \leftarrow (A_{0-3})$ p=4–7

Example: Move data in accumulator to ports 4 and 5.

```
OUTP45: MOVD P4,A        ;MOVE ACC BITS 0-3 TO PORT 4
        SWAP A           ;EXCHANGE ACC BITS 0-3 AND 4-7
        MOVD P5,A        ;MOVE ACC BITS 0-3 TO PORT 5
```

MOVP A,@A Move Current Page Data to Accumulator

Opcode:

1	0	1	0	0	0	1	1

This is a 2-cycle instruction. The contents of the program memory location addressed by the accumulator are moved to the accumulator. Only bits 0–7 of the program counter are affected, limiting the program memory reference to the current page. The program counter is restored following this operation.
(A) ← ((A))

Note: This is a 1-byte, 2-cycle instruction. If it appears in location 255 of a program memory page, @A addresses a location in the following page.

Example:

```
MOV128: MOV   A,#128      ;MOVE '128' DEC TO ACC
        MOVP A,@A         ;CONTENTS OF 129TH LOCATION
                          ;IN CURRENT PAGE ARE MOVED TO
                          ;ACC
```

MOVP3 A,@A Move Page 3 Data to Accumulator

Opcode:

1	1	1	0	0	0	1	1

This is a 2-cycle instruction. The contents of the program memory location within page 3, addressed by the accumulator, are moved to the accumulator. The program counter is restored following this operation.
(A) ← ((A)) within page 3

Example: Look up ASCII equivalent of hexadecimal code in table contained at the beginning of page 3. Note that ASCII characters are designated by a 7-bit code; the eighth bit is always reset.

```
TABSCH: MOV A,#OB8H       ;MOVE 'B8' HEX TO ACC (10111000)
        ANL A,#7FH        ;LOGICAL AND ACC TO MASK BIT
                          ;7 (00111000)
        MOVP3, A,@A       ;MOVE CONTENTS OF LOCATION
                          ;'38' HEX IN PAGE 3 TO ACC
                          ;(ASCII '8')
```

Access contents of location in page 3 labelled TAB1. Assume current program location is not in page 3.

```
TABSCH: MOV A,#TAB1       ;ISOLATE BITS 0–7
                          ;OF LABEL
                          ;ADDRESS VALUE
        MOVP3 A,@A        ;MOVE CONTENT OF PAGE 3
                          ;LOCATION LABELED 'TAB1'
                          ;TO ACC
```

NOP The NOP Instruction

Opcode:

0	0	0	0	0	0	0	0

No operation is performed. Execution continues with the following instruction.

ORL A,Rr Logical OR Accumulator With Register Mask

Opcode:

0	1	0	0	1	r_2	r_1	r_0

Data in the accumulator is logically ORed with the mask contained in working register 'r'.
(A) ← (A) OR (Rr) r=0–7

Example:

```
ORREG: ORL A,R4           ;'OR' ACC CONTENTS WITH
                          ;MASK IN REG 4
```

ORL A,@Rr Logical OR Accumulator With Memory Mask

Opcode: | 0 1 0 0 | 0 0 0 r |

Data in the accumulator is logically ORed with the mask contained in the data memory location referenced by register 'r', bits 0–5.
(A) ← (A) OR ((Rr)) r=0–1

Example:
```
ORDM: MOVE R0,#3FH    ;MOVE '3F' HEX TO REG 0
      ORL A,@R0       ;'OR' ACC CONTENTS WITH MASK
                      ;IN LOCATION 63
```

ORL A,#data Logical OR Accumulator With Immediate Mask

Opcode: | 0 1 0 0 | 0 0 1 1 | • | d_7 d_6 d_5 d_4 | d_3 d_2 d_1 d_0 |

This is a 2-cycle instruction. Data in the accumulator is logically ORed with an immediately-specified mask.
(A) ← (A) OR data

Example:
```
ORID: ORL A,#'X'      ;'OR' ACC CONTENTS WITH MASK
                      ;01011000 (ASCII VALUE OF 'X')
```

ORL Pp,#data Logical OR Port 1–2 With Immediate Mask

Opcode: | 1 0 0 0 | 1 0 p_1 p_0 | • | d_7 d_6 d_5 d_4 | d_3 d_2 d_1 d_0 |

This is a 2-cycle instruction. Data on port 'p' is logically ORed with an immediately-specified mask.
(Pp) ← (Pp) OR data p=1–2 (see OUTL instruction)

Example:
```
ORP1: ORL P1,#0FFH    ;'OR' PORT 1 CONTENTS WITH
                      ;MASK 'FF' HEX (SET PORT 1
                      'TO ALL ONES)
```

ORLD Pp,A Logical OR Port 4–7 With Accumulator Mask

Opcode: | 1 0 0 0 | 1 1 p_1 p_0 |

This is a 2-cycle instruction. Data on 8243 port 'p' is logically ORed with the digit mask contained in accumulator bits 0–3,
(Pp) (Pp) OR (A_{0-3}) p=4–7 (See MOVD instruction)

Example:
```
ORP7: ORLD P7,A       ;'OR' PORT 7 CONTENTS
                      ;WITH ACC BITS 0–3
```

OUT DBB,A Output Accumulator Contents to Data Bus Buffer

Opcode: | 0 0 0 0 | 0 0 1 0 |

Contents of the accumulator are transferred to the Data Bus Buffer Output register and the Output Buffer Full (OBF) flag is set to one.
(DBB) ← (A)
OBF ← 1

Example:
```
OUTDBB: OUT DBB,A     ;OUTPUT THE CONTENTS OF
                      ;THE ACC TO DBBOUT
```

OUTL Pp,A Output Accumulator Data to Port 1 and 2

Opcode:

0	0	1	1	1	0	p_1	p_0

This is a 2-cycle instruction. Data residing in the accumulator is transferred (written) to port 'p' and latched.
(Pp) ← (A) P=1–2

Note: Bits 0–1 of the opcode are used to represent PORT 1 and PORT 2. If you are coding in binary rather than assembly language, the mapping is as follows:

Bits	p1	p0	Port
	0	0	X
	0	1	1
	1	0	2
	1	1	X

Example:

```
OUTLP: MOV A,R7      ;MOVE REG 7 CONTENTS TO ACC
       OUTL P2,A     ;OUTPUT ACC CONTENTS TO PORT2
       MOV A,R6      ;MOVE REG 6 CONTENTS TO ACC
       OUTL P1,A     ;OUTPUT ACC CONTENTS TO PORT 1
```

RET Return Without PSW Restore

Opcode:

1	0	0	0	0	0	1	1

This is a 2-cycle instruction. The stack pointer (PSW bits 0–2) is decremented. The program counter is then restored from the stack. PSW bits 4–7 are not restored.
(SP) ← (SP) – 1
(PC) ← ((SP))

RETR Return With PSW Restore

Opcode:

1	0	0	1	0	0	1	1

This is a 2-cycle instruction. The stack pointer is decremented. The program counter and bits 4–7 of the PSW are then restored from the stack. Note that RETR should be used to return from an interrupt, but should not be used within the interrupt service routine as it signals the end of an interrupt routine.
(SP) ← (SP) – 1
(PC) ← ((SP))
(PSW_{4-7}) ← ((SP))

RL A Rotate Left Without Carry

Opcode:

1	1	1	0	0	1	1	1

The contents of the accumulator are rotated left one bit. Bit 7 is rotated into the bit 0 position.
$(A_{n+1}) \leftarrow (A_n)$ n=0–6
$(A_0) \leftarrow (A_7)$

Example: Assume accumulator contains 10110001.

```
RLNC: RL A           ;NEW ACC CONTENTS ARE 01100011
```

RLC A Rotate Left Through Carry

Opcode:

1	1	1	1	0	1	1	1

The contents of the accumulator are rotated left one bit. Bit 7 replaces the carry bit; the carry bit is rotated into the bit 0 position.

$(A_{n+1}) \leftarrow (A_n)$ n=0–6
$(A_0) \leftarrow (C)$
$(C) \leftarrow (A7)$

Example: Assume accumulator contains a 'signed' number; isolate sign without changing value.

```
RLTC: CLR C        ;CLEAR CARRY TO ZERO
      RLC A        ;ROTATE ACC LEFT, SIGN
                   ;BIT (7) IS PLACED IN CARRY
      RR A         ;ROTATE ACC RIGHT — VALUE
                   ;(BITS 0-6) IS RESTORED,
                   ;CARRY UNCHANGED, BIT 7
                   ;IS ZERO
```

RR A Rotate Right Without Carry

Opcode:

0	1	1	1	0	1	1	1

The contents of the accumulator are rotated right one bit. Bit 0 is rotated into the bit 7 position.

$(A_n) \leftarrow (A_{n+1})$ n=0–6
$(A_7) \leftarrow (A_0)$

Example: Assume accumulator contains 10110001.

```
RRNC: RRA          ;NEW ACC CONTENTS ARE 11011000
```

RRC A Rotate Right Through Carry

Opcode:

0	1	1	0	0	1	1	1

The contents of the accumulator are rotated right one bit. Bit 0 replaces the carry bit; the carry bit is rotated into the bit 7 position.

$(A_n) \leftarrow (A_{n+1})$ n=0–6
$(A_7) \leftarrow (C)$
$(C) \leftarrow (A_0)$

Example: Assume carry is not set and accumulator contains 10110001.

```
RRTC: RRCA         ;CARRY IS SET AND ACC
                   ;CONTAINS 01011000
```

SEL RB0 Select Register Bank 0

Opcode:

1	1	0	0	0	1	0	1

PSW BIT 4 is set to zero. References to working registers 0–7 address data memory locations 0–7. This is the recommended setting for normal program execution.

$(BS) \leftarrow 0$

SEL RB1 Select Register Bank 1

Opcode:

1	1	0	1	0	1	0	1

PSW bit 4 is set to one. References to working registers 0–7 address data memory locations 24–31. This is the recommended setting for interrupt service routines, since locations 0–7 are left intact. The setting of PSW bit 4 in effect at the time of an interrupt is restored by the RETR instruction when the interrupt service routine is completed.

(BS) ← 1

Example: Assume an IBF interrupt has occurred, control has passed to program memory location 3, and PSW bit 4 was zero before the interrupt.

```
LOC3: JMP INIT          ;JUMP TO ROUTINE 'INIT'
                .
                .
                .
INIT: MOV R7,A          ;MOV ACC CONTENTS TO
                        ;LOCATION 7
      SEL RB1           ;SELECT REG BANK 1
      MOV R7,#OFAH      ;MOVE 'FA' HEX TO LOCATION 31
                .
                .
                .
      SEL RB0           ;SELECT REG BANK 0
      MOV A,R7          ;RESTORE ACC FROM LOCATION 7
      RETR              ;RETURN—RESTORE PC AND PSW
```

STOP TCNT Stop Timer/Event Counter

Opcode:

0	1	1	0	0	1	0	1

This instruction is used to stop both time accumulation and event counting.

Example: Disable interrupt, but jump to interrupt routine after eight overflows and stop timer. Count overflows in register 7.

```
START: DIS TCNTI        ;DISABLE TIMER INTERRUPT
       CLR A            ;CLEAR ACC TO ZERO
       MOV T,A          :MOV ZERO TO TIMER
       MOV R7,A         :MOVE ZERO TO REG 7
       STRT T           ;START TIMER
MAIN: JTF COUNT         ;JUMP TO ROUTINE 'COUNT'
                        ;IF TF=1 AND CLEAR TIMER FLAG
      JMP MAIN          ;CLOSE LOOP
COUNT: INC R7           ;INCREMENT REG 7
       MOV A,R7         ;MOVE REG 7 CONTENTS TO ACC
       JB3 INT          ;JUMP TO ROUTINE 'INT' IF ACC
                        ;BIT 3 IS SET (REG 7=8)
       JMP MAIN         ;OTHERWISE RETURN TO ROUTINE
                        ;MAIN
                .
                .
                .
INT: STOP TCNT          ;STOP TIMER
     JMP 7H             ;JUMP TO LOCATION 7 (TIMER
                        ;INTERRUPT ROUTINE)
```

STRT CNT Start Event Counter

Opcode:

0	1	0	0	0	1	0	1

The TEST 1 (T_1) pin is enabled as the event-counter input and the counter is started. The event-counter register is incremented with each high to low transition on the T_1 pin.

Example: Initialize and start event counter. Assume overflow is desired with first T_1 input.

```
STARTC: EN TCNTI        ;ENABLE COUNTER INTERRUPT
        MOV A,#OFFH     ;MOVE 'FF' HEX (ONES) TO
                        ;ACC
        MOV T,A         ;MOVE ONES TO COUNTER
        STRT CNT        ;INPUT AND START
```

STRT T Start Timer

Opcode:

0	1	0	1	0	1	0	1

Timer accumulation is initiated in the timer register. The register is incremented every 32 instruction cycles. The prescaler which counts the 32 cycles is cleared but the timer register is not.

Example: Initialize and start timer.

```
STARTT: EN TCNTI        ;ENABLE TIMER INTERRUPT
        CLR A           :CLEAR ACC TO ZEROS
        MOV T,A         ;MOVE ZEROS TO TIMER
        STRT T          ;START TIMER
```

SWAP A Swap Nibbles Within Accumulator

Opcode:

0	1	0	0	0	1	1	1

Bits 0–3 of the accumulator are swapped with bits 4–7 of the accumulator.

$(A_{4-7}) \longleftrightarrow (A_{0-3})$

Example: Pack bits 0–3 of locations 50–51 into location 50.

```
PCKDIG: MOV R0,#50      ;MOVE '50' DEC TO REG 0
        MOV R1,#51      ;MOVE '51' DEC TO REG 1
        XCHD A,@R0      ;EXCHANGE BIT 0–3 OF ACC
                        ;AND LOCATION 50
        SWAP A          ;SWAP BITS 0–3 AND 4–7 OF ACC
        XCHD A,@R1      ;EXCHANGE BITS 0–3 OF ACC AND
                        ;LOCATION 51
        MOV @R0,A       ;MOVE CONTENTS OF ACC TO
                        ;LOCATION 51
```

XCH A,Rr Exchange Accumulator-Register Contents

Opcode:

0	0	1	0	1	r_2	r_1	r_0

The contents of the accumulator and the contents of working register 'r' are exchanged.

$(A) \longleftrightarrow (Rr)$ $\quad r=0–7$

Example: Move PSW contents to Reg 7 without losing accumulator contents.

```
XCHAR7: XCH A,R7        ;EXCHANGE CONTENTS OF REG 7
                        ;AND ACC
        MOV A,PSW       ;MOVE PSW CONTENTS TO ACC
        XCH A,R7        ;EXCHANGE CONTENTS OF REG 7
                        ;AND ACC AGAIN
```

XCH A,@Rr Exchange Accumulator and Data Memory Contents

Opcode:

0	0	1	0	0	0	0	r

The contents of the accumulator and the contents of the data memory location addressed by bits 0–5 of register 'r' are exchanged. Register 'r' contents are unaffected.

(A) ⟷ ((Rr)) r=0–1

Example: Decrement contents of location 52.

```
DEC52: MOV R0,#52      ;MOVE '52' DEC TO ADDRESS
                       ;REG 0
       XCH A,@R0       ;EXCHANGE CONTENTS OF ACC
                       ;AND LOCATION 52
       DEC A           ;DECREMENT ACC CONTENTS
       XCH A,@R0       ;EXCHANGE CONTENTS OF ACC
                       ;AND LOCATION 52 AGAIN
```

XCHD A,@Rr Exchange Accumulator and Data Memory 4-bit Data

Opcode:

0	0	1	1	0	0	0	r

This instruction exchanges bits 0–3 of the accumulator with bits 0–3 of the data memory location addressed by bits 0–5 of register 'r'. Bits 4–7 of the accumulator, bits 4–7 of the data memory location, and the contents of register 'r' are unaffected.

$(A_{0-3}) \longleftrightarrow ((Rr_{0-3}))$ r=0–1

Example: Assume program counter contents have been stacked in locations 22–23.

```
XCHNIB: MOV R0,#23     ;MOVE '23' DEC TO REG 0
        CLR A          ;CLEAR ACC TO ZEROS
        XCHD A,@R0     ;EXCHANGE BITS 0-3 OF ACC
                       ;AND LOCATION 23 (BITS 8-11
                       ;OF PC ARE ZEROED, ADDRESS
                       ;REFERS TO PAGE 0)
```

XRL A,Rr Logical XOR Accumulator With Register Mask

Opcode:

1	1	0	1	1	r_2	r_1	r_0

Data in the accumulator is EXCLUSIVE ORed with the mask contained in working register 'r'.

(A) ← (A) XOR (Rr) r=0–7

Example:

```
XORREG: XRL A,R5       ;'XOR' ACC CONTENTS WITH
                       ;MASK IN REG 5
```

XRL A,@Rr Logical XOR Accumulator With Memory Mask

Opcode:

1	1	0	1	0	0	0	r

Data in the accumulator is EXCLUSIVE ORed with the mask contained in the data memory location addressed by register 'r', bits 0–5.

(A) ← (A) XOR ((Rr)) r=0–1

Example:

```
XORDM: MOV R1,#20H     ;MOVE '20' HEX TO REG 1
       XRL A,@R1       ;'XOR' ACC CONTENTS WITH MASK
                       ;IN LOCATION 32
```

XRL A,#data Logical XOR Accumulator With Immediate Mask

Opcode:

1	1	0	1	0	0	1	1	•	d_7	d_6	d_5	d_4	d_3	d_2	d_1	d_0

This is a 2-cycle instruction. Data in the accumulator is EXCLUSIVE ORed with an immediately-specified mask.

(A) ← (A) XOR data

Example:

```
XORID: XOR A,#HEXTEN      ;XOR CONTENTS OF ACC WITH
                          ;MASK EQUAL VALUE OF SYMBOL
                          ;'HEXTEN'
```

CHAPTER 4
SINGLE-STEP, PROGRAMMING, AND POWER-DOWN MODES

SINGLE-STEP

The UPI family has a single-step mode which allows the user to manually step through his program one instruction at a time. While stopped, the address of the next instruction to be fetched is available on PORT 1 and the lower 2 bits of PORT 2. The single-step feature simplifies program debugging by allowing the user to easily follow program execution.

Figure 4-1 illustrates a recommended circuit for single-step operation, while Figure 4-2 shows the timing relationship between the SYNC output and the $\overline{SS}$ input. During single-step operation, PORT 1 and part of PORT 2 are used to output address information. In order to retain the normal I/O functions of PORTS 1 and 2, a separate latch can be used as shown in Figure 4-3.

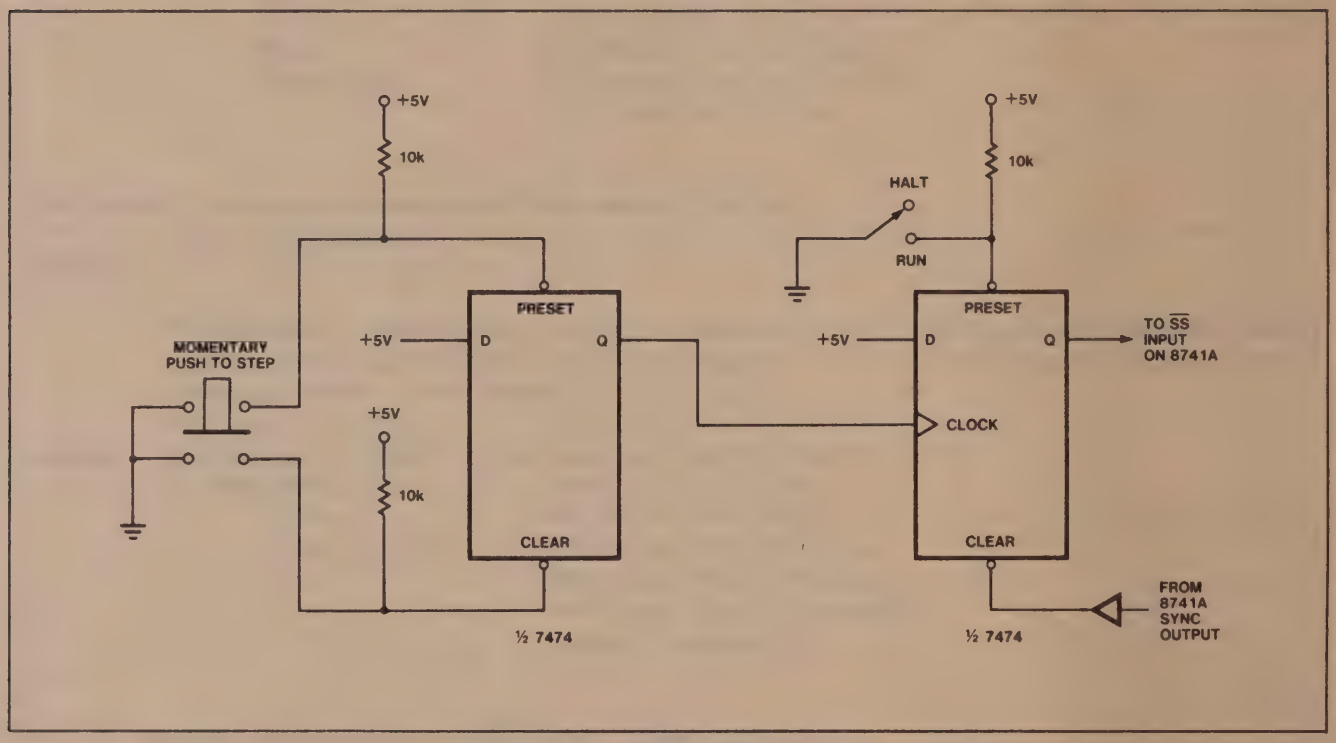


Figure 4-1. Single-Step Circuit

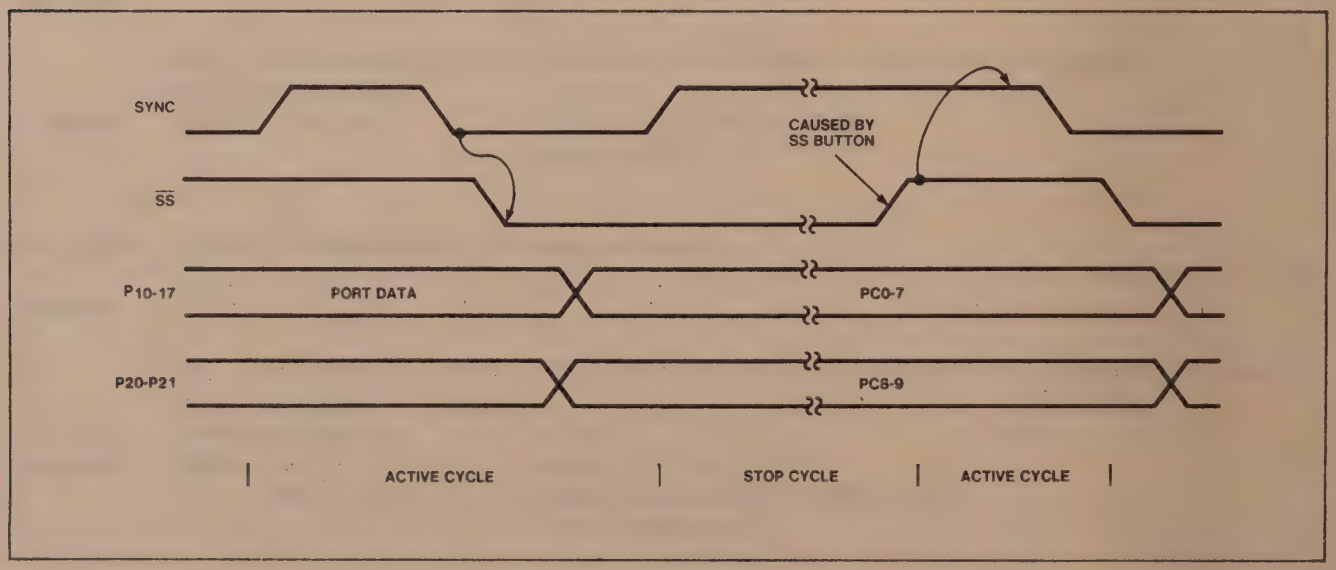


Figure 4-2. Single-Step Timing

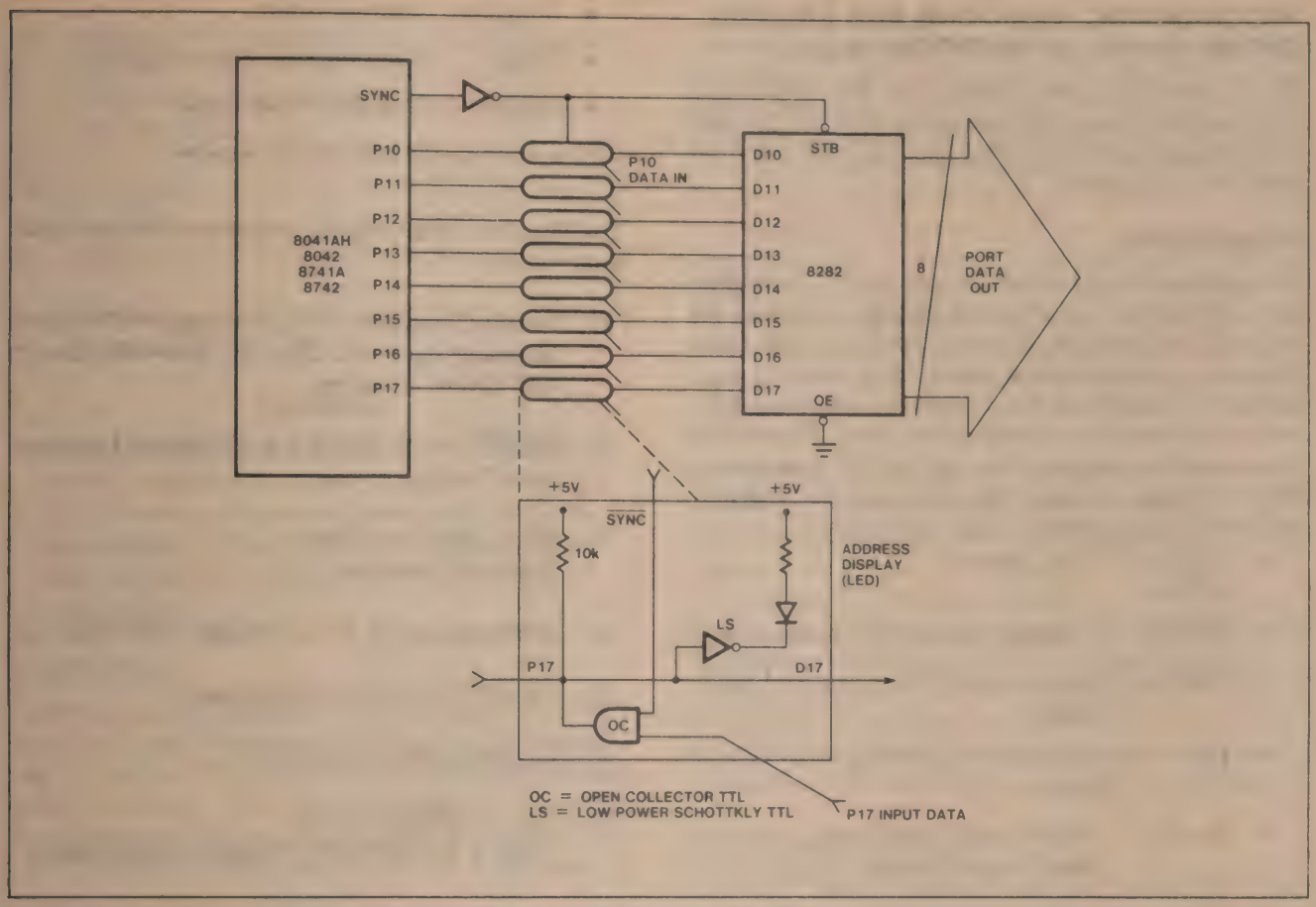


Figure 4-3. Latching Port Data

Timing

The sequence of single-step operation is as follows:

1) The processor is requested to stop by applying a low level on $\overline{SS}$. The $\overline{SS}$ input should not be brought low while SYNC is high. (The UPI samples the $\overline{SS}$ pin in the middle of the SYNC pulse).

2) The processor responds to the request by stopping during the instruction fetch portion of the next instruction. If a double cycle instruction is in progress when the single-step command is received, both cycles will be completed before stopping.

3) The processor acknowledges it has entered the stopped state by raising SYNC high. In this state, which can be maintained indefinitely, the 10-bit address of the next instruction to be fetched is present on PORT 1 and the lower 2 bits of PORT 2.

4) $\overline{SS}$ is then raised high to bring the processor out of the stopped mode allowing it to fetch the next instruction. The exit from stop is indicated by the processor bringing SYNC low.

5) To stop the processor at the next instruction $\overline{SS}$ must be brought low again before the next SYNC pulse—the circuit in Figure 4-1 uses the trailing edge of the previous pulse. If $\overline{SS}$ is left high, the processor remains in the "RUN" mode.

Figure 4-1 shows a schematic for implementing single-step. A single D-type flip-flop with preset and clear is used to generate $\overline{SS}$. In the RUN mode $\overline{SS}$ is held high by keeping the flip-flop preset (preset has precedence over the clear input). To enter single-step, preset is removed allowing SYNC to bring $\overline{SS}$ low via the clear input. Note that SYNC must be buffered since the SN7474 is equivalent to 3 TTL loads.

The processor is now in the stopped state. The next instruction is initiated by clocking "1" into the flip-flop. This "1" will not appear on $\overline{SS}$ unless SYNC is high (i.e., clear must be removed from the flip-flop). In response to $\overline{SS}$ going high, the processor begins an instruction fetch which brings SYNC low. $\overline{SS}$ is then reset through the clear input and the processor again enters the stopped state.

PROGRAMMING, VERIFYING AND ERASING EPROM (8741A, 8742 EPROM ONLY)

The internal Program Memory of the 8741A and 8742 may be erased and reprogrammed by the user as explained in the following sections. See the data sheet for more detail.

Programming

The programming procedure consists of the following: activating the program mode, applying an address, latching the address, applying data, and applying a programming pulse. Each word is programmed completely before moving on to the next and is followed by a verification step. Figure 4-4 illustrates the programming and verifying sequence. The following is a list of the pins used for programming and a description of their functions:

- XTAL 1, XTAL 2 — Clock Input
- $\overline{RESET}$ — Initialization and Address Latching
- TEST 0 — Selection of Program or Verify Mode
- EA — Activation of Program/Verify Modes
- D_0–D_7 — Address and Data Input; Data Output During Verify
- P_{20}, P_{21} — Address Input
- V_{DD} — Programming Power Supply
- PROG — Program Pulse Input

NOTE: All set-up and hold times are 4 cycles.

The detailed Programming sequence (for one byte) is as follows:

1) Initial Conditions: $V_{CC} = V_{DD} = 5V$; Clock Running; $A_0 = 0V$, $\overline{CS} = 5V$; EA = 5V; D_0–D_7 and PROG Floating.

2) $\overline{RESET} = 0V$, TEST 0 = 0V (Select Programming Mode).

3) EA = 23V for 8741A
   EA = 18V for 8742

4) Address applied to D_0–D_7 and PORTS 20–22.

5) $\overline{RESET} = 5V$ (Latch Address).

6) Data applied to D_0–D_7.

7) $V_{DD} = 25V$ for 8741A
   $V_{DD} = 21V$ for 8742 (Programming Power).

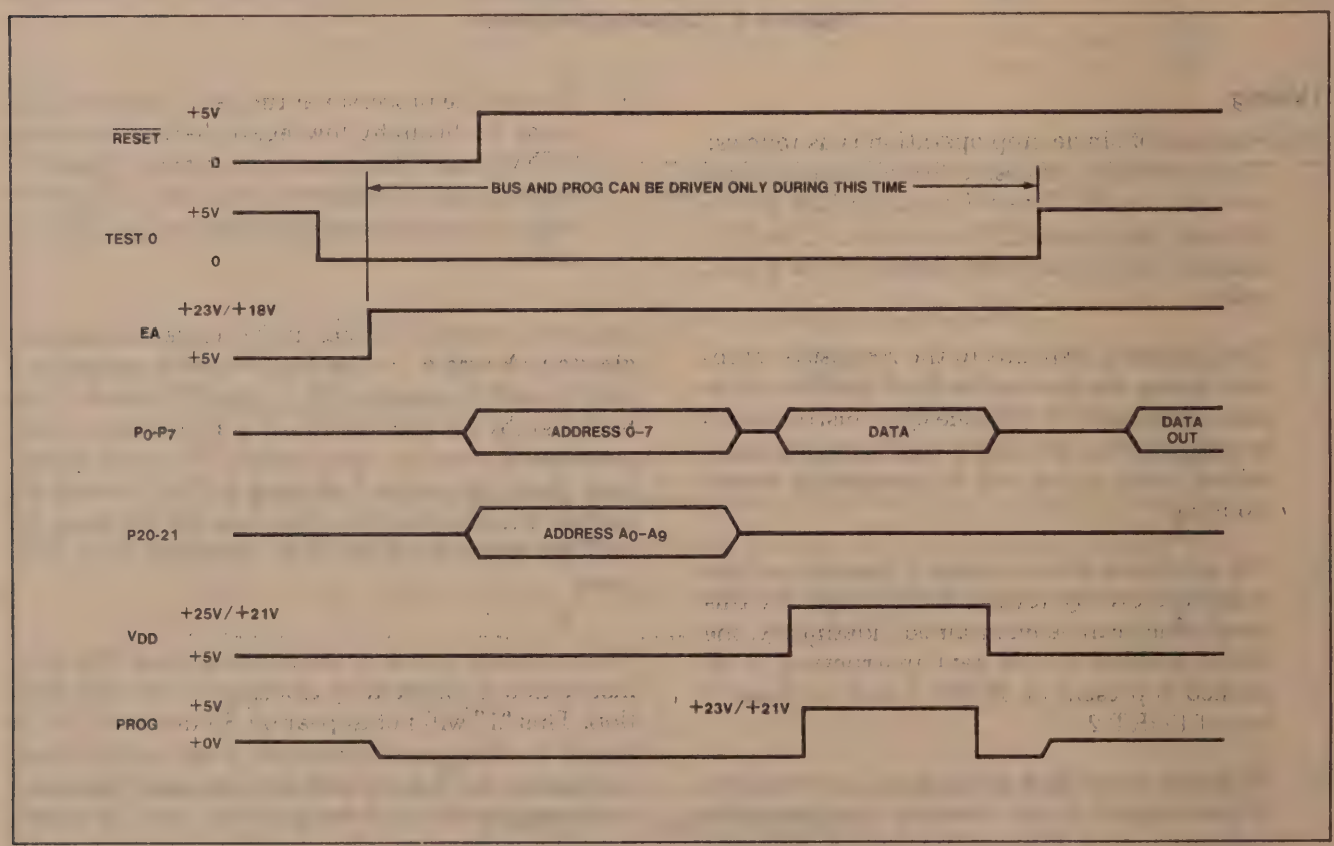


Figure 4-4. Programming Sequence

8) PROG = 0V followed by one 50 msec pulse of 23V for 8741A
PROG = 0V followed by one 50 msec pulse of 18V for 8742.

9) V_{DD} = 5V.

10) TEST 0 = 5V (Select Verify Mode).

11) Read data on D_0–D_7 and verify EPROM cell contents.

WARNING

An attempt to program a mis-socketed 8741A or 8742 will result in severe damage to the part. An indication of a properly socketed part is the appearance of the SYNC clock output. The lack of this clock may be used to disable the programmer.

Verification

Verification is accomplished by latching in an address as in the Programming Mode and then applying "1" to the TEST 0 input. The word stored at the selected address then appears on the D_0–D_7 lines. Note that verification can be applied to both ROM's and EPROM's independently of the programming procedure. See the data sheet.

The detailed Verifying sequence (for one byte) is as follows:

1) Initial Conditions: V_{CC} = V_{DD} = 5V; Clock Running; A_0 = 0V, $\overline{CS}$ = 5V; EA = 5V; D_0–D_7 and PROG Floating.

2) $\overline{RESET}$ = 0V, TEST 0 = 5V (Verify Mode).

3) EA = 23V for 8741A
EA = 18V for 8742

4) Address applied to D_0–D_7 and PORTS 20–22.

5) $\overline{RESET}$ = 5V (Latch Address)

6) Read data on D_0–D_7 and verify EPROM cell contents.

Erasing

The program memory of the 8741A or 8742 may be erased to zeros by exposing its translucent lid to shortwave ultraviolet light.

EPROM Light Sensitivity

The erasure characteristics of the 8741A or 8742 EPROM are such that erasure begins to occur when exposed to light with wavelengths shorter than approximately 4000 Angstroms. It should be noted that sunlight and certain types of fluorescent lamps have wavelengths in the 3000–4000 Angstrom range. Data shows that constant exposure to room level fluorescent lighting could erase the typical 8741A in approximately 3 years while it would take approximately 1 week to cause erasure when exposed to direct sunlight. If the 8741A or 8742 is to be exposed to these types of lighting conditions for extended periods of time, opaque labels (available from Intel) should be placed over the 8741A or 8742 window to prevent unintentional erasure.

The recommended erasure procedure for the 8741A or 8742 is exposure to shortwave ultraviolet light which has a wavelength of 2537 Angstroms. The integrated dose (i.e., UV intensity × exposure time) for erasure should be a minimum of 15W-sec/cm^2 power rating. The erasure time with this dosage is approximately 15 minutes using an ultraviolet lamp with a 12,000 μW/cm^2 power rating. The 8741A or 8742 should be placed within 1 inch of the lamp tubes during erasure. Some lamps have a filter on their tubes which should be removed before erasure.

EXTERNAL ACCESS

The UPI family has an External Access mode (EA) which puts the processor into a test mode. This mode allows the user to disable the internal program memory and execute from external memory. External Access mode is useful in testing because it allows the user to test the processor's functions directly. It is only useful for testing since this mode uses D_0–D_7, PORTS 10–17 and PORTS 20–22.

This mode is invoked by connecting the EA pin to 5V. The 11-bit current program counter contents then come out on PORTS 10–17 and PORTS 20–22 after the SYNC output goes high. (PORT 10 is the least significant bit.) The desired instruction opcode is placed on D_0–D_7 before the start of state S_1. During state S_1, the opcode is sampled from D_0–D_7 and subsequently executed in place of the internal program memory contents.

The program counter contents are multiplexed with the I/O port data on PORTS 10–17 and PORTS 20–22. The I/O port data may be demultiplexed using an external latch on the rising edge of SYNC. The program counter contents may be demultiplexed similarly using the trailing edge of SYNC.

Reading and/or writing the Data Bus Buffer registers is still allowed although only when D_0–D_7 are not being sampled for opcode data. In practice, since this sampling time is not known externally, reads or

writes on the system bus are done during SYNC high time. Approximately 600ns are available for each read or write cycle.

POWER DOWN MODE (8041AH/8042 ROM ONLY)

Extra circuitry is included in the ROM version to allow low-power, standby operation. Power is removed from all system elements except the internal data RAM in the low-power mode. Thus the contents of RAM can be maintained and the device draws only 10 to 15% of its normal power.

The V_{CC} pin serves as the 5V power supply pin for all of the ROM version's circuitry except the data RAM array. The V_{DD} pin supplies only the RAM array. In normal operation, both V_{CC} and V_{DD} are connected to the same 5V power supply.

To enter the Power-Down mode, the $\overline{RESET}$ signal to the UPI is asserted. This ensures the memory will not be inadvertently altered by the UPI during power-down. The V_{CC} pin is then grounded while V_{DD} is maintained at 5V. Figure 4-5 illustrates a recommended Power-Down sequence. The sequence typically occurs as follows:

1) Imminent power supply failure is detected by user defined circuitry. The signal must occur early enough to guarantee the 8041AH or 8042 can save all necessary data before V_{CC} falls outside normal operating tolerance.

2) A "Power Failure" signal is used to interrupt the processor (via a timer overflow interrupt, for instance) and call a Power Failure service routine.

3) The Power Failure routine saves all important data and machine status in the RAM array. The routine may also initiate transfer of a backup supply to the V_{DD} pin and indicate to external circuitry that the Power Failure routine is complete.

4) A $\overline{RESET}$ signal is applied by external hardware to guarantee data will not be altered as the power supply falls out of limits. $\overline{RESET}$ must be low until V_{CC} reaches ground potential.

Recovery from the Power-Down mode can occur as any other power-on sequence. An external 1 μfd capacitor on the $\overline{RESET}$ input will provide the necessary initialization pulse.

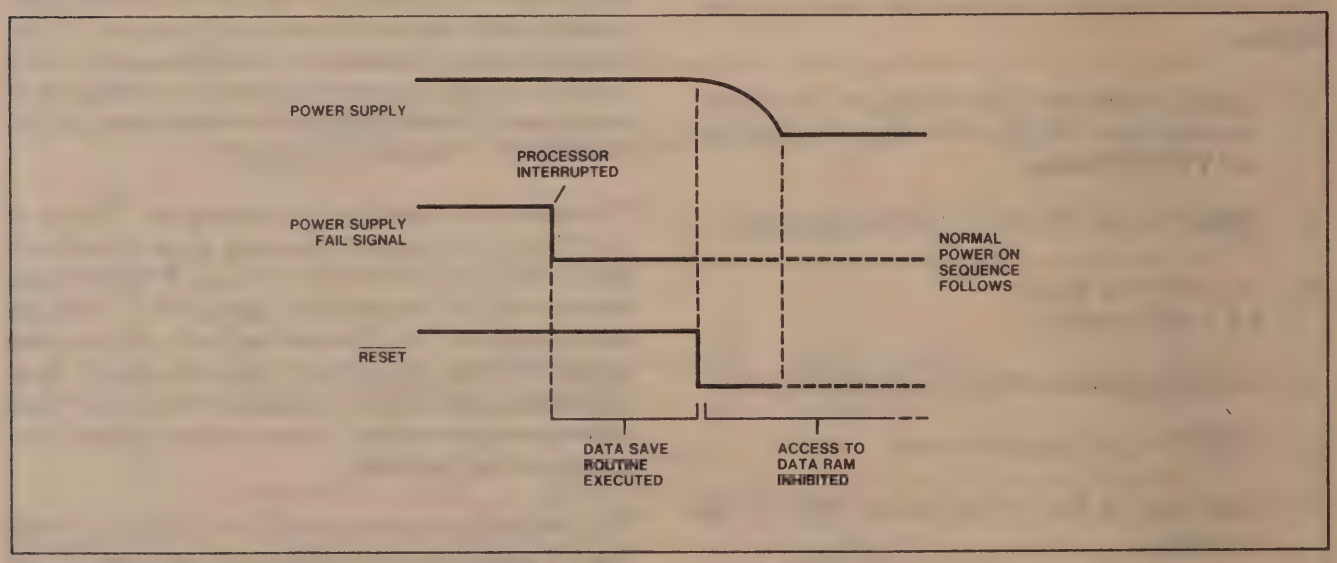


Figure 4-5. Power-Down Sequence

CHAPTER 5
SYSTEM OPERATION

BUS INTERFACE

The UPI-41AH, 42 Microcomputer functions as a peripheral to a master processor by using the data bus buffer registers to handle data transfers. The DBB configuration is illustrated in Figure 5-1. The UPI-41AH, 42 Microcomputer's 8 three-state data lines (D_7–D_0) connect directly to the master processor's data bus. Data transfer to the master is controlled by 4 external inputs to the UPI:

- A_0 Address Input signifying command or data
- $\overline{CS}$ Chip Select
- $\overline{RD}$ Read strobe
- $\overline{WR}$ Write strobe

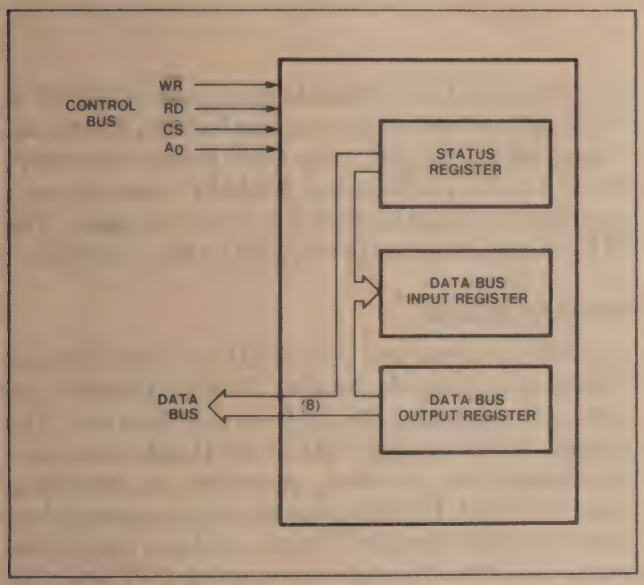


Figure 5-1. Data Bus Register Configuration

The master processor addresses the UPI-41AH, 42 Microcomputer as a standard peripheral device. Table 5-1 shows the conditions for data transfer:

Table 5-1. Data Transfer Controls

$\overline{CS}$	A_0	$\overline{RD}$	$\overline{WR}$	Condition
0	0	0	1	Read DBBOUT
0	1	0	1	Read STATUS
0	0	1	0	Write DBBIN data, set $F_1 = 0$
0	1	1	0	Write DBBIN command set $F_1 = 1$
1	x	x	x	Disable DBB

Reading the DBBOUT Register

The sequence for reading the DBBOUT register is shown in Figure 5-2. This operation causes the 8-bit contents of the DBBOUT register to be placed on the system Data Bus. The OBF flag is cleared automatically.

Reading STATUS

The sequence for reading the UPI-41AH, 42 Microcomputer's 8 STATUS bits is shown in Figure 5-3. This operation causes the 8-bit STATUS register contents to be placed on the system Data Bus as shown.

Write Data to DBBIN

The sequence for writing data to the DBBIN register is shown in Figure 5-4. This operation causes the system Data Bus contents to be transferred to the DBBIN register and the IBF flag is set. Also, the F_1 flag is cleared ($F_1 = 0$) and an interrupt request is generated. When the IBF interrupt is enabled, a jump to location 3 will occur. The interrupt request is cleared upon entering the IBF service routine or by a system RESET input.

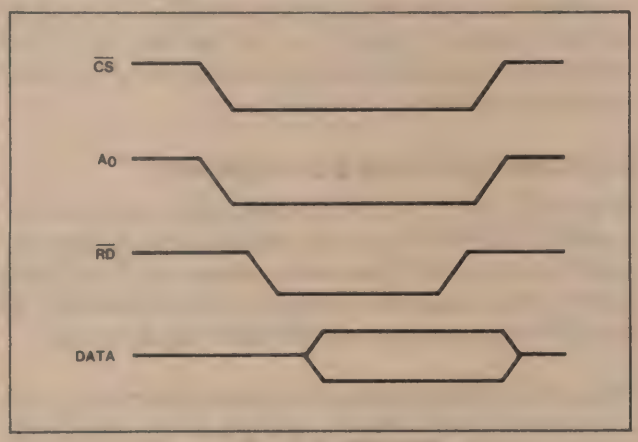


Figure 5-2. DBBOUT Read

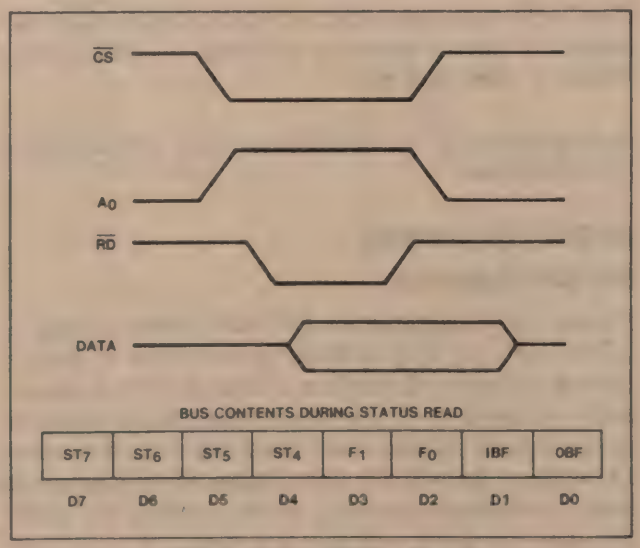


Figure 5-3. Status Read

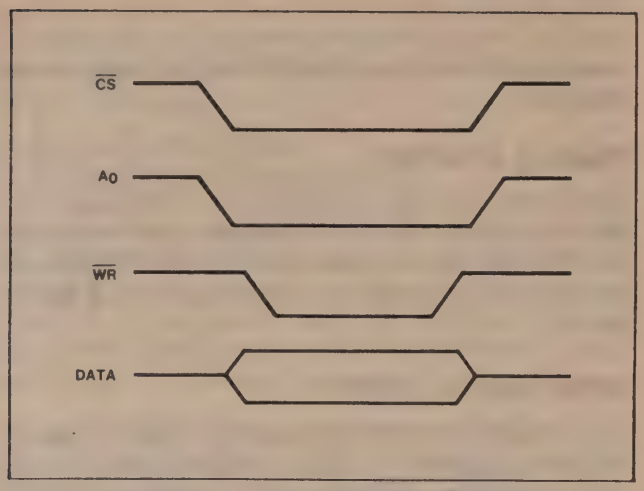


Figure 5-4. Writing Data to DBBIN

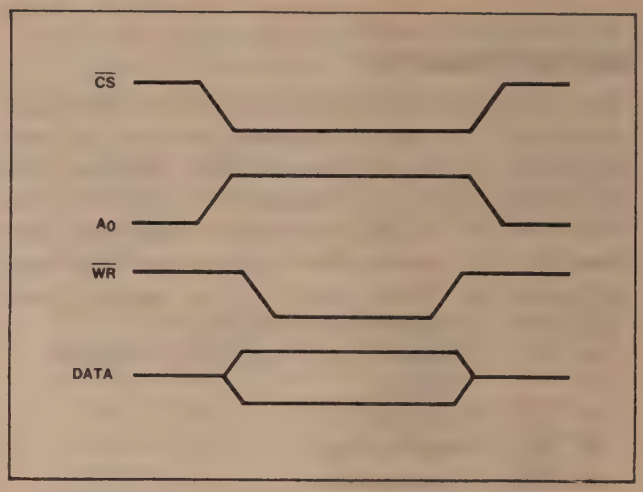


Figure 5-5. Writing Commands to DBBIN

Writing Commands to DBBIN

The sequence for writing commands to the DBBIN register is shown in Figure 5-5. This sequence is identical to a data write except that the A_0 input is latched in the F_1 flag ($F_1 = 1$). The IBF flag is set and an interrupt request is generated when the master writes a command to DBB.

Operations of Data Bus Registers

The UPI-41AH, 42 Microcomputer controls the transfer of DBB data to its accumulator by executing INput and OUTput instructions. An IN A,DBB instruction causes the contents to be transferred to the UPI accumulator and the IBF flag is cleared.

The OUT DBB,A instruction causes the contents of the accumulator to be transferred to the DBBOUT register. The OBF flag is set.

The UPI's data bus buffer interface is applicable to a variety of microprocessors including the 8086, 8088, 8085AH, 8080, and 8048.

A description of the interface to each of these processors follows.

DESIGN EXAMPLES

8085AH Interface

Figure 5-6 illustrates an 8085AH system using a UPI-41AH, 42. The 8085AH system uses a multiplexed address and data bus. During I/O the 8 upper address lines (A_8–A_{15}) contain the same I/O address as the lower 8 address/data lines (A_0–A_7); therefore I/O address decoding is done using only the upper 8 lines to eliminate latching of the address. An 8205 decoder provides address decoding for both the UPI-41AH, 42 and the 8237. Data is transferred using the two DMA handshaking lines of PORT 2. The 8237 performs the actual bus transfer operation. Using the UPI-41AH, 42's OBF master interrupt, the UPI-41A notifies the 8085AH upon transfer completion using the RST 5.5 interrupt input. The $\overline{IBF}$ master interrupt is not used in this example.

8088 Interface

Figure 5-7 illustrates a UPI-41AH, 42 interface to an 8088 minimum mode system. Two 8-bit latches are used to demultiplex the address and data bus. The address bus is 20-lines wide. For I/O only, the lower 16 address lines are used, providing an addressing range of 64K. UPI address selection is accomplished using an 8205 decoder. The A_0 address line of the bus is connected to the corresponding UPI input for register selection. Since the UPI-41A is polled by the 8088, neither DMA nor master interrupt capabilities of the UPI-41AH, 42 are used in the figure.

8086 Interface

The UPI-41AH, 42 can be used on an 8086 maximum mode system as shown in figure 5-8. The address and data bus is demultiplexed using three 8282 latches providing separate address and data buses. The address bus is 20-lines wide and the data bus is 16-lines wide. Multiplexed control lines are decoded by the 8288. The UPI's $\overline{CS}$ input is provided by linear selection. Note that the UPI-41AH, 42 is both I/O mapped and memory mapped as a result of the linear addressing technique. An address decoder may be used to limit the UPI-41AH, 42 to a specific I/O mapped address. Address line A_1 is connected to the UPI's A_0 input. This insures that the registers of the UPI will have even I/O addresses. Data will be transferred on D_0–D_7 lines only. This allows the I/O registers to be accessed using byte manipulation instructions.

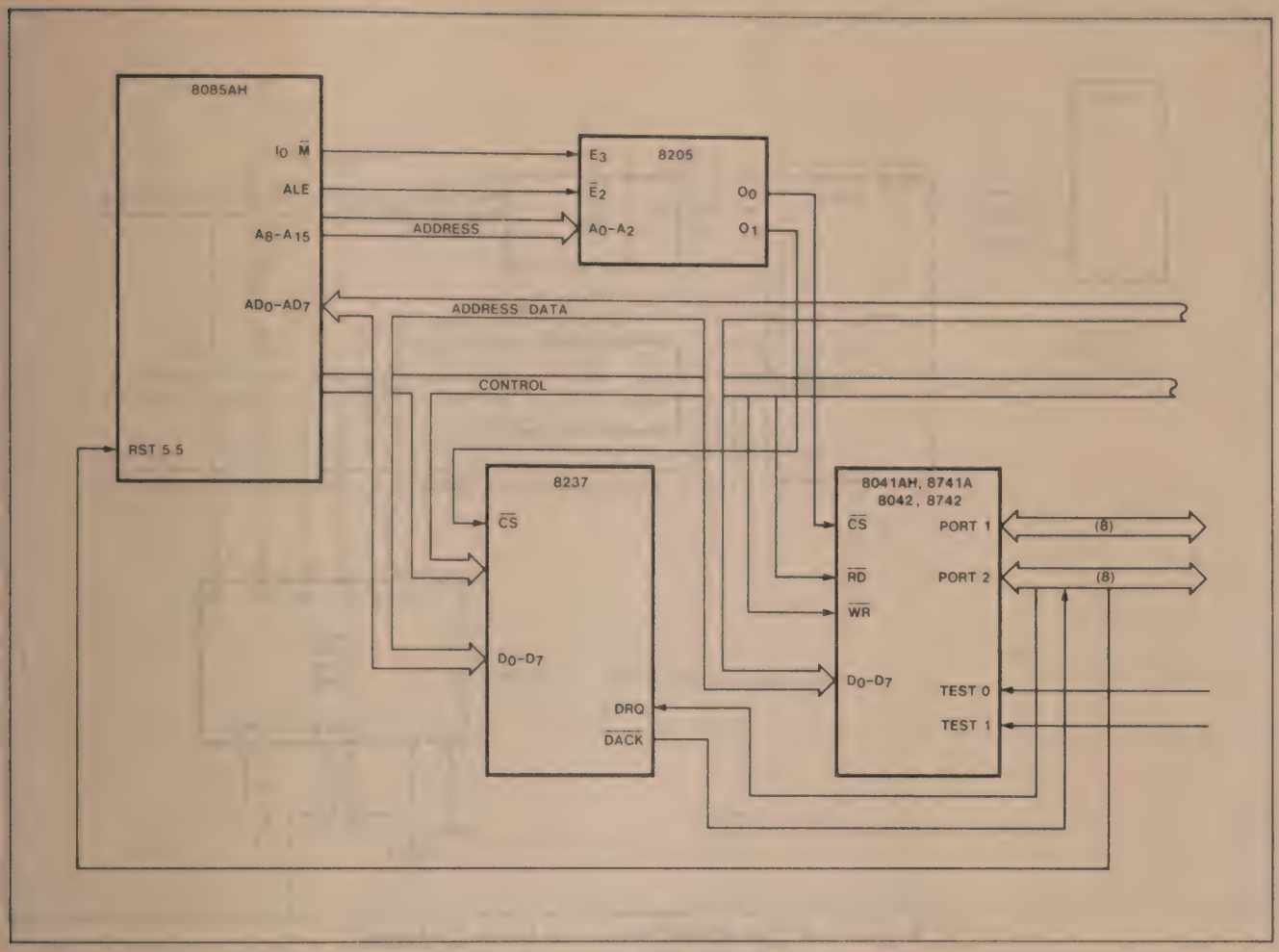


Figure 5-6. 8085AH-UPI System

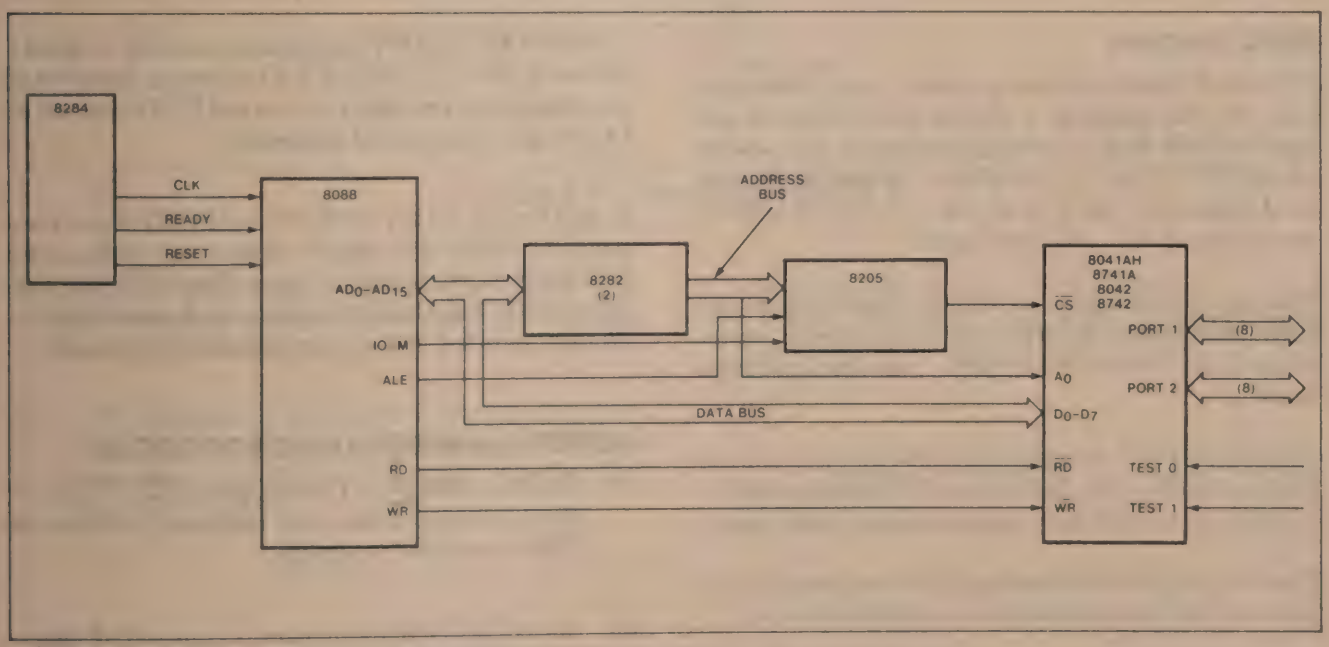


Figure 5-7. 8088-UPI Minimum Mode System

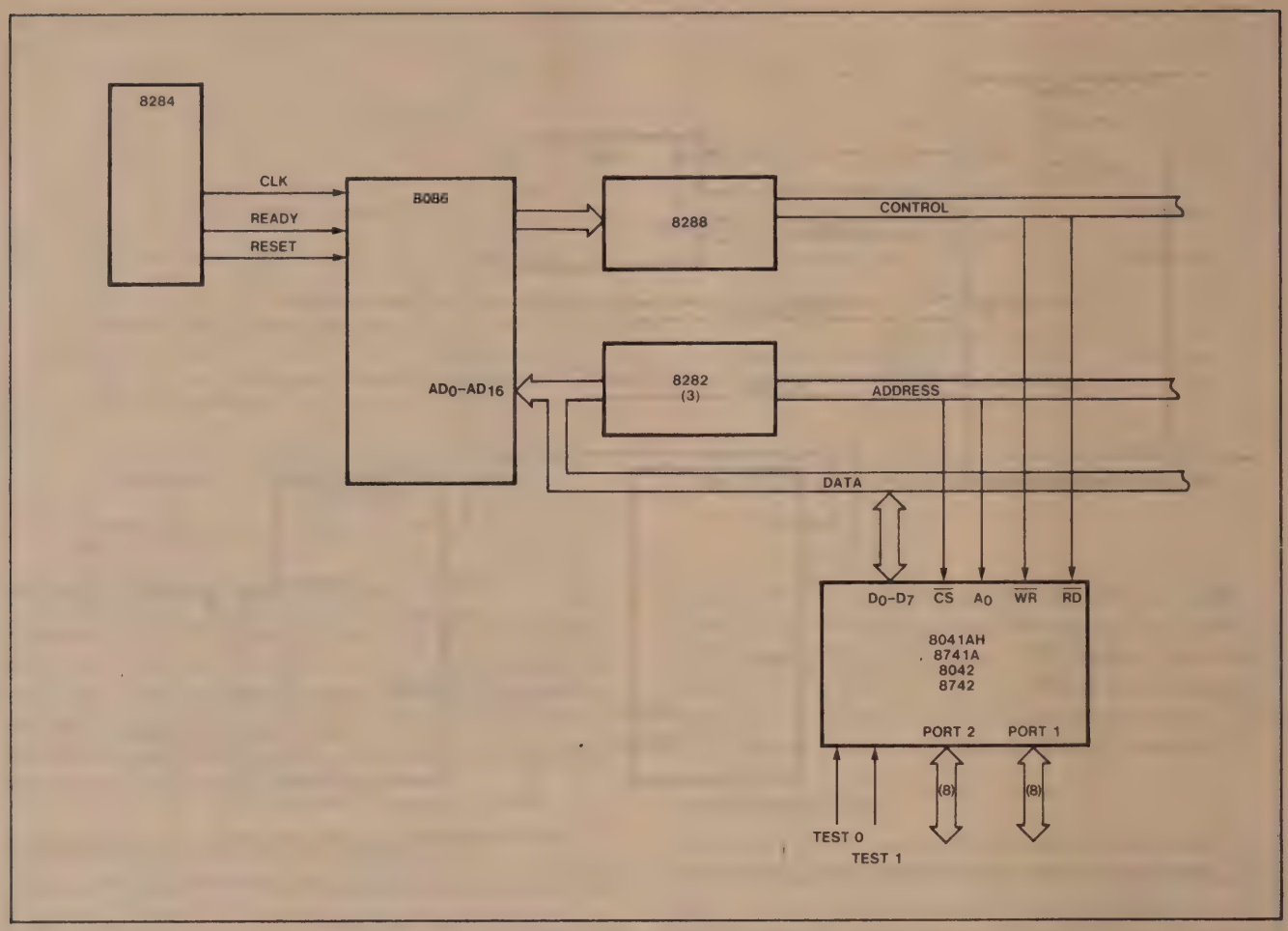


Figure 5-8. 8086-UPI Maximum Mode Systems

8080 Interface

Figure 5-9 illustrates the interface to an 8080A system. In this example, a crystal and capacitor are used for UPI-41AH, 42 timing reference and power-on RESET. If the 2-MHz 8080A 2-phase clock were used instead of the crystal, the UPI-41AH, UPI-42 would run at only 16% full speed.

The A_0 and $\overline{CS}$ inputs are direct connections to the 8080 address bus. In larger systems, however, either of these inputs may be decoded from the 16 address lines.

The $\overline{RD}$ and $\overline{WR}$ inputs to the UPI can be either the $\overline{IOR}$ and $\overline{IOW}$ or the $\overline{MEMR}$ and $\overline{MEMR}$ signals depending on the I/O mapping technique to be used.

The UPI can be addressed as an I/O device using INput and OUTput instructions in 8080 software.

8048 Interface

Figure 5-10 shows the UPI interface to an 8048 master processor.

The 8048 $\overline{RD}$ and $\overline{WR}$ outputs are directly compatible with the UPI. Figure 5-11 shows a distributed processing system with up to seven UPI's connected to a single 8048 master processor.

In this configuration the 8048 uses PORT 0 as a data bus. I/O PORT 2 is used to select one of the seven UPI's when data transfer occurs. The UPI's are programmed to handle isolated tasks and, since they operate in parallel, system throughput is increased.

GENERAL HANDSHAKING PROTOCOL

1) Master reads STATUS register ($\overline{RD}$, $\overline{CS}$, A_0 = (0, 0, 1)) in polling or in response to either an $\overline{IBF}$ or an OBF interrupt.

2) If the UPI DBBIN register is empty (IBF flag = 0), Master writes a word to the DBBIN register ($\overline{WR}$, $\overline{CS}$, A_0 = (0, 0, 1) or (0, 0, 0)). If $A_0 = 1$, write command word, set F_1. If $A_0 = 0$, write data word, $F_1 = 0$.

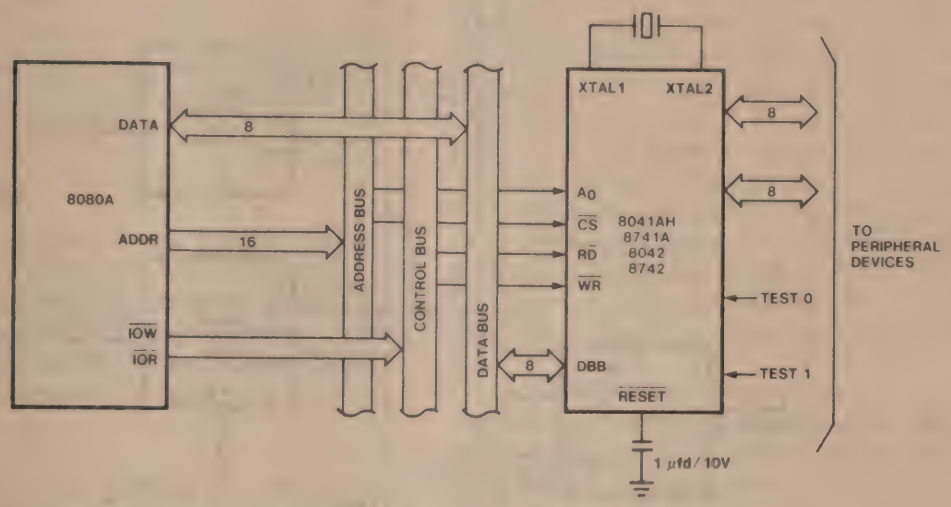


Figure 5-9. 8080A-UPI Interface

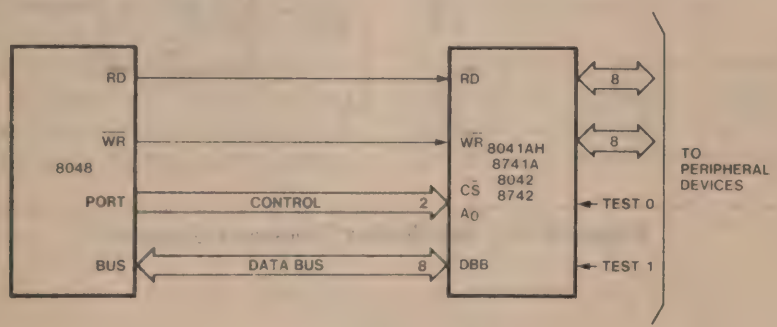


Figure 5-10. 8048-UPI Interface

3) If the UPI DBBOUT register is full (OBF flag = 1), Master reads a word from the DBBOUT register ($\overline{RD}$, $\overline{CS}$, A_0 = (0, 0, 0)).

4) UPI recognizes IBF (via IBF interrupt or JNIBF). Input data or command word is processed, depending on F_1; IBF is reset. Repeat step 1 above.

5) UPI-41AH, 42 recognizes OBF flag = 0 (via JOBF). Next word is output to DBBOUT register, OBF is set. Repeat step 1 above.

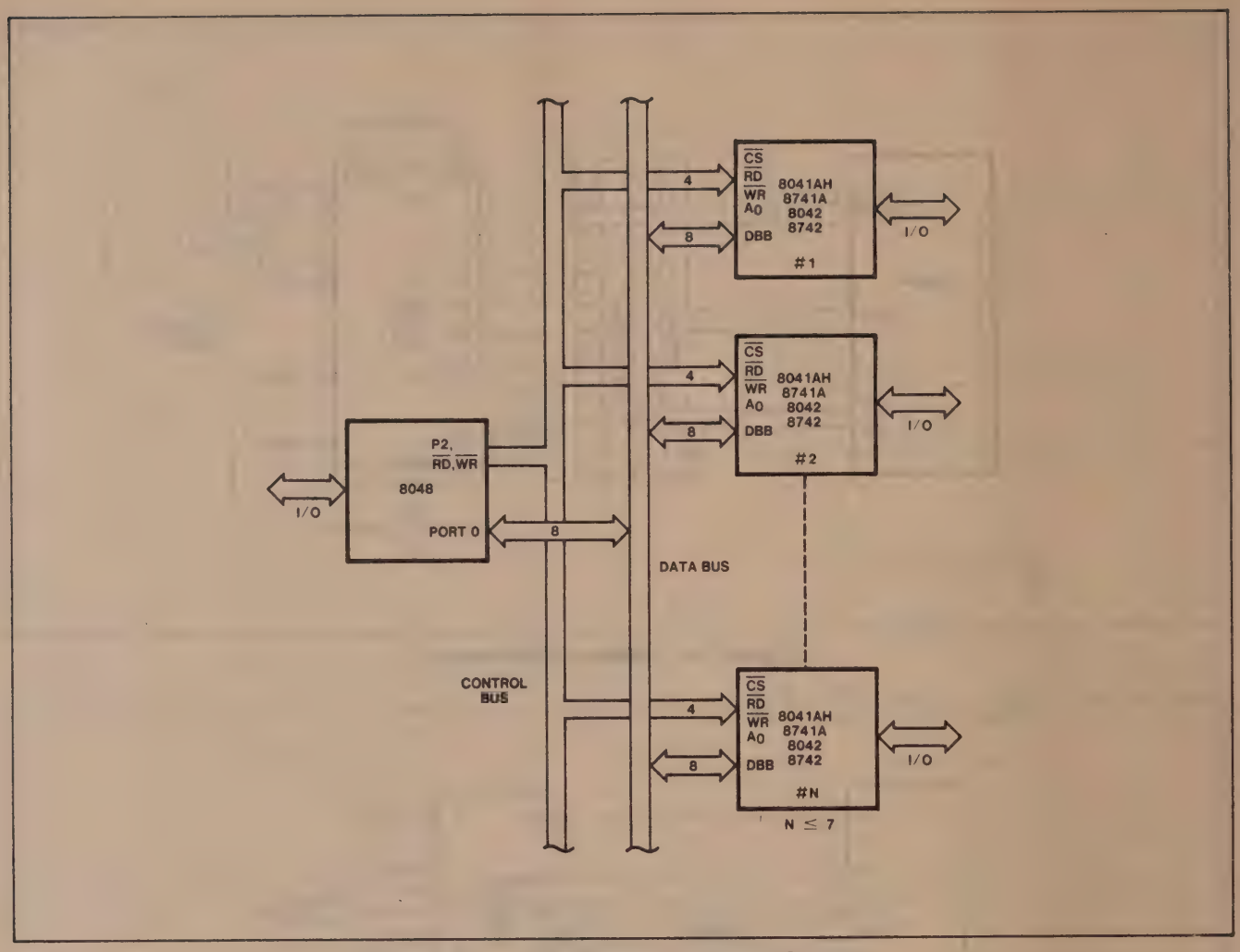


Figure 5-11. Distributed Processor System

Chapter 6
APPLICATIONS

ABSTRACTS

The UPI-41A is designed to fill a wide variety of low to medium speed peripheral interface applications where flexibility and easy implementation are important considerations. The following examples illustrate some typical applications.

Keyboard Encoder

Figure 6-1 illustrates a keyboard encoder configuration using the UPI and the 8243 I/O expander to scan a 128-key matrix. The encoder has switch matrix scanning logic, N-key rollover logic, ROM look-up table, FIFO character buffer, and additional outputs for display functions, control keys or other special functions.

PORT 1 and PORTs 4–7 provide the interface to the keyboard. PORT 1 lines are set one at a time to select the various key matrix rows.

When a row is energized, all 16 columns (i.e., PORTs 4–7 inputs) are sampled to determine if any switch in the row is closed. The scanning software is code efficient because the UPI instruction set includes individual bit set/clear operations and expander PORTs 4–7 can be directly addressed with single, 2-byte instructions. Also, accumulator bits can be tested in a single operation. Scan time for 128 keys is about 10 ms. Each matrix point has a unique binary code which is used to address ROM when a key closure is detected. Page 3 of ROM contains a look-up table with useable codes (i.e., ASCII, EBCDIC, etc.) which correspond to each key. When a valid key closure is detected the ROM code corresponding to that key is stored in a FIFO buffer in data memory for transfer to the master processor. To avoid stray noise and switch bounce, a key closure must be detected on two consecutive scans before it is considered valid and loaded into the FIFO buffer. The FIFO buffer allows multiple keys to be processed as they are depressed without regard to when they are released, a condition known as N-key rollover.

The basic features of this encoder are fairly standard and require only about 500 bytes of memory. Since the UPI is programmable and has additional memory capacity it can handle a number of other functions. For example, special keys can be programmed to give an entry on closing as well as opening. Also, I/O lines are available to control a 16-digit, 7-segment display. The UPI can also be programmed to recognize special combinations of characters such as commands, then transfer only the decoded information to the master processor.

A complete keyboard application has been developed for the UPI-41A. A description is included in this section. The code for the application is available in the Intel Insite Library (program AB 147).

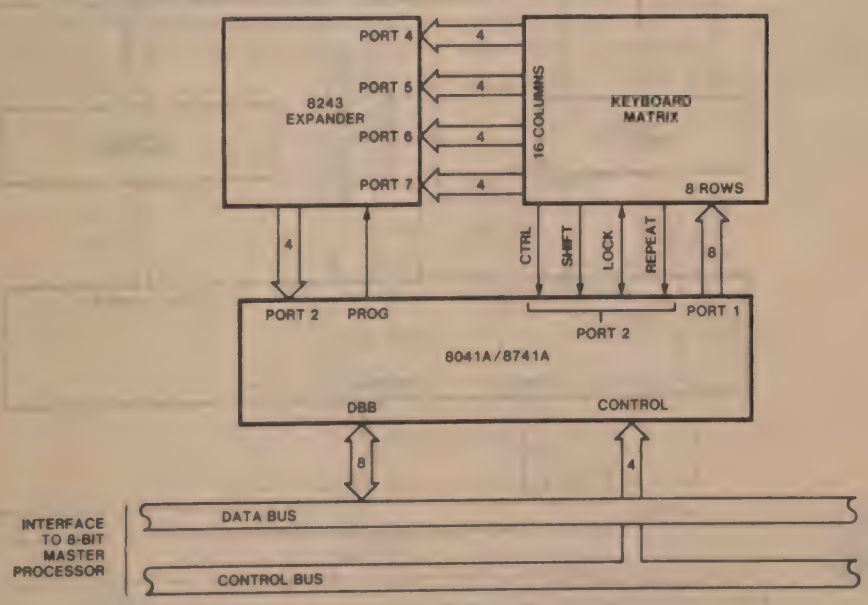


Figure 6-1. Keyboard Encoder Configuration

Matrix Printer Interface

The matrix printer interface illustrated in Figure 6-2 is a typical application for the UPI-41A. The actual printer mechanism could be any of the numerous dot-matrix types and similar configurations can be shown for drum, spherical head, daisy wheel or chain type printers.

The bus structure shown represents a generalized, 8-bit system bus configuration. The UPI's three-state interface port and asynchronous data buffer registers allow it to connect directly to this type of system for efficient, two-way data transfer.

The UPI's two on-board I/O ports provide up to 16 input and output signals to control the printer mechanism. The timer/event counter is used for generating a timing sequence to control print head position, line feed, carriage return, and other sequences. The on-board program memory provides character generation for 5 × 7, 7 × 9, or other dot matrix formats. As an added feature a portion of the 64 × 8-bit data memory can be used as a FIFO buffer so that the master processor can send a block of data at a high rate. The UPI can then output characters from the buffer at a rate the printer can accept while the master processor returns to other tasks.

The 8295 Printer Controller is an example of an 8041A preprogrammed as a dot matrix printer interface.

Tape Cassette Controller

Figure 6-3 illustrates a digital cassette interface which can be implemented with the UPI-41A. Two sections of the tape transport are controlled by the UPI: digital data/command logic, and motor servo control.

The motor servo requires a speed reference in the form of a monostable pulse whose width is proportional to the desired speed. The UPI monitors a prerecorded clock from the tape and uses its on-board interval timer to generate the required speed reference pulses at each clock transition.

Recorded data from the tape is supplied serially by the data/command logic and is converted to 8-bit words by the UPI, then transferred to the master processor. At 10 ips tape speed the UPI can easily handle the 8000 bps data rate. To record data, the UPI uses the two input lines to the data/command logic which control the flux direction in the recording head. The UPI also monitors 4 status lines from the tape transport including: end of tape, cassette

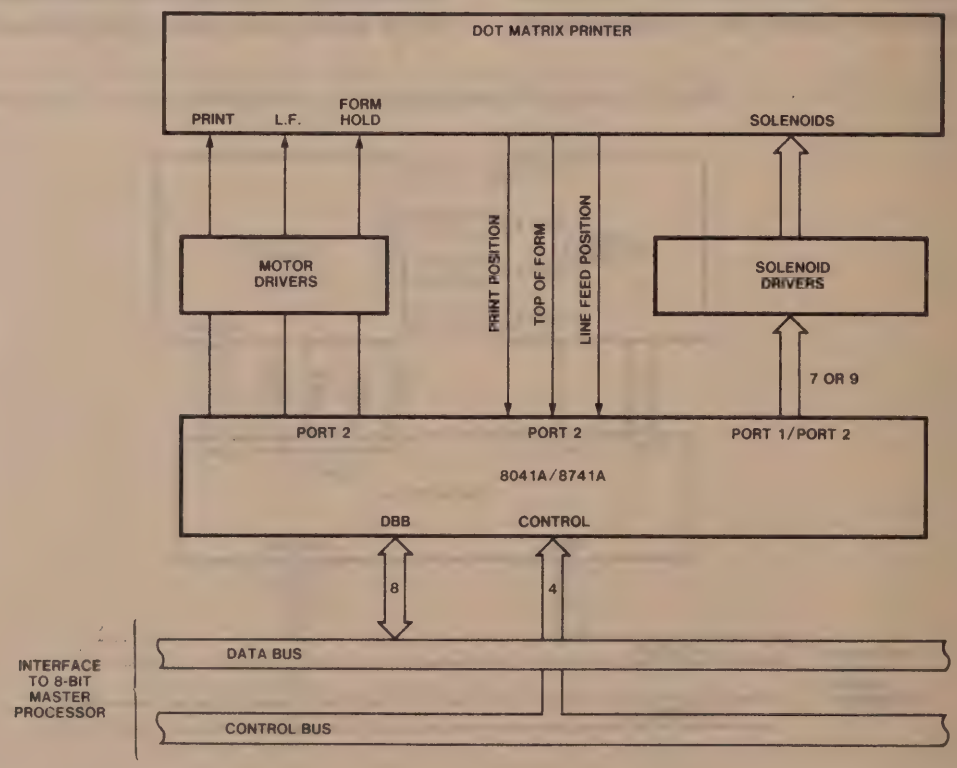


Figure 6-2. Matrix Printer Controller

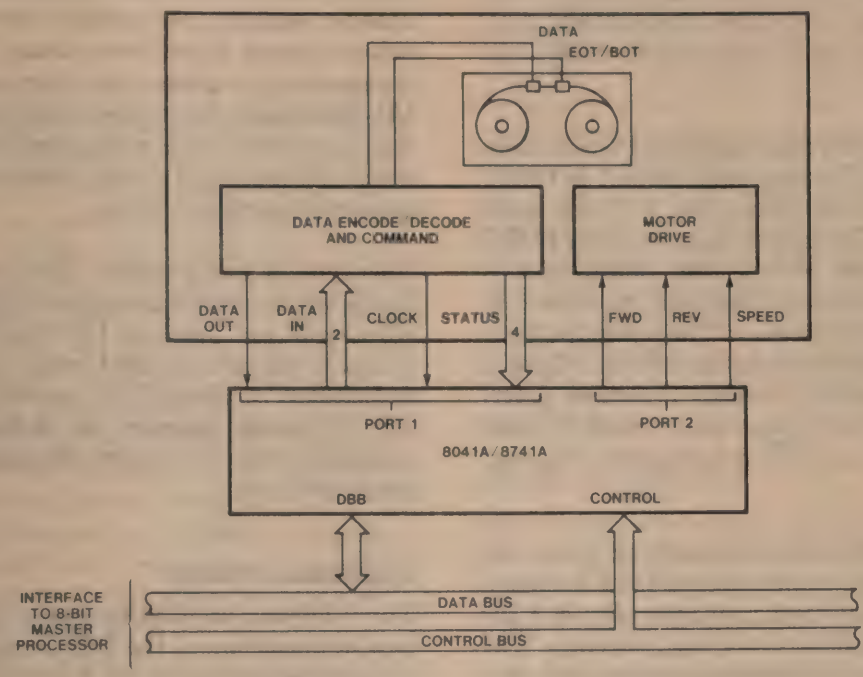


Figure 6-3. Tape Transport Controller

inserted, busy, and write permit. All control signals can be handled by the UPI's two I/O ports.

Universal I/O Interface

Figure 6-4 shows an I/O interface design based on the UPI. This configuration includes 12 parallel I/O lines and a serial (RS232C) interface for full duplex data transfer up to 1200 baud. This type of design can be used to interface a master processor to a broad spectrum of peripheral devices as well as to a serial communication channel.

PORT 1 is used strictly for I/O in this example while PORT 2 lines provide five functions:

- P_{23}-P_{20} I/O lines (bidirectional)
- P_{24} Request to send (RTS)
- P_{25} Clear to Send (CTS)
- P_{26} Interrupt to master
- P_{27} Serial data out

The parallel I/O lines make use of the bidirectional port structure of the UPI. Any line can function as an input or output. All port lines are automatically initialized to 1 by a system $\overline{\text{RESET}}$ pulse and remain

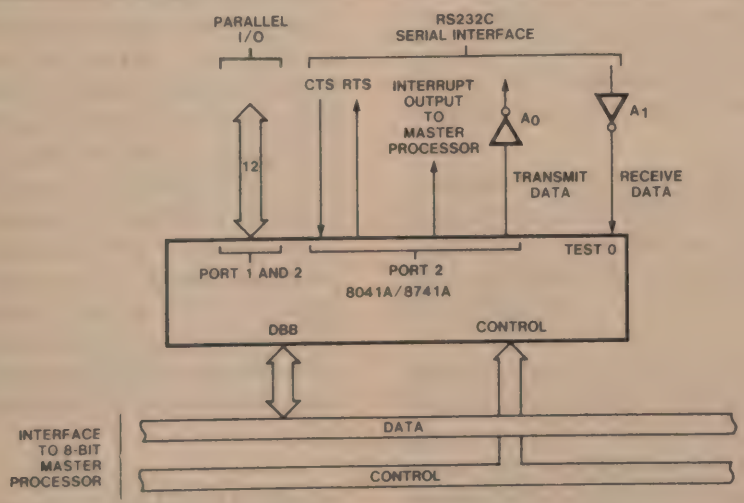


Figure 6-4. Universal I/O Interface

latched. An external TTL signal connected to a port line will override the UPI's 50K-ohm internal pull-up so that an INPUT instruction will correctly sample the TTL signal.

Four PORT 2 lines function as general I/O similar to PORT 1. Also, the RTS signal is generated on PORT 2 under software control when the UPI has serial data to send. The CTS signal is monitored via PORT 2 as an enable to the UPI to send serial data. A PORT 2 line is also used as a software generated interrupt to the master processor. The interrupt functions as a service request when the UPI has a byte of data to transfer or when it is ready to receive. Alternatively, the EN FLAGS instruction could be used to create the OBF and $\overline{IBF}$ interrupts on P_{24} and P_{25}.

The RS232C interface is implemented using the TEST 0 pin as a receive input and a PORT 2 pin as a transmit output. External packages (A_0, A_1) are used to provide RS232C drive requirements. The serial receive software is interrupt driven and uses the on-chip timer to perform time critical serial control. After a start bit is detected the interval timer can be preset to generate an interrupt at the proper time for sampling the serial bit stream. This eliminates the need for software timing loops and allows the processor to proceed to other tasks (i.e., parallel I/O operations) between serial bit samples. Software flags are used so the main program can determine when the interrupt driven receive program has a character assembled for it.

This type of configuration allows system designers flexibility in designing custom I/O interfaces for specific serial and parallel I/O applications. For instance, a second or third serial channel could be substituted in place of the parallel I/O if required. The UPI's data memory can buffer data and commands for up to 4 low-speed channels (110 baud teletypewriter, etc.)

Application Notes

The following application notes illustrate the various applications of the UPI family. Other related publications including the *8048 Family Application Handbook* are available through the Intel Literature Department.

INTRODUCTION TO THE UPI-41A™

Introduction

Since the introduction in 1974 of the second generation of microprocessors, such as the 8080, a wide range of peripheral interface devices have appeared. At first, these devices solved application problems of a general nature; i.e., parallel interface (8255), serial interface (8251), timing (8253), interrupt control (8259). However, as the speed and density of LSI technology increased, more and more intelligence was incorporated into the peripheral devices. This allowed more specific application problems to be solved, such as floppy disk control (8271), CRT control (8275), and data link control (8273). The advantage to the system designer of this increased peripheral device intelligence is that many of the peripheral control tasks are now handled externally to the main processor in the peripheral hardware rather than internally in the main processor software. This reduced main processor overhead results in increased system throughput and reduced software complexity.

In spite of the number of peripheral devices available, the pervasiveness of the microprocessor has been such that there is still a large number of peripheral control applications not yet satisfied by dedicated LSI. Complicating this problem is the fact that new applications are emerging faster than the manufacturers can react in developing new, dedicated peripheral controllers. To address this problem, a new microcomputer-based Universal Peripheral Interface (UPI-41A) device was developed.

In essence, the UPI-41A acts as a slave processor to the main system CPU. The UPI contains its own processor, memory, and I/O, and is completely user programmable; that is, the entire peripheral control algorithm can be programmed locally in the UPI, instead of taxing the master processor's main memory. This distributed processing concept allows the UPI to handle the real-time tasks such as encoding keyboards, controlling printers, or multiplexing displays, while the main processor is handling non-real-time dependent tasks such as buffer management or arithmetic. The UPI relies on the master only for initialization, elementary commands, and data transfers. This technique results in an overall increase in system efficiency since both processors—the master CPU and the slave UPI—are working in parallel.

This application note presents three UPI-41A applications which are roughly divided into two groups: applications whose complexity and UPI code space requirements allow them to either stand alone or be incorporated as just one task in a "multi-tasking" UPI, and applications which are complete UPI applications in themselves. Applications in the first group are a simple LED display and sensor matrix controllers. A combination serial/parallel/ I/O device is an application in the second group. Each application illustrates different UPI configurations and features. However, before the application details are presented, a section on the UPI/master protocol requirements is included. These protocol requirements are key to UPI software development. For convenience, the UPI block diagram is reproduced in Figure 1 and the instruction set summary in Table 1.

UPI-41 vs. UPI-41A

The UPI-41A is an enhanced version of the UPI-41. It incorporates several architectural features not found on the "non-A" device:

- Separate Data In and Data Out data bus buffer registers
- User-definable STATUS register bits
- Programmable master interrupts for the OBF and $\overline{\text{IBF}}$ flags
- Programmable DMA interface to external DMA controller.

The separate Data In (DBBIN) and Data Out (DBBOUT) registers greatly simplify the master/UPI protocol compated to the UPI-41. The master need only check IBF before writing to DBBIN and OBF before reading DBBOUT. No data bus buffer lock-out is required.

The most significant nibble of the STATUS register, undefined in the UPI-41, is user-definable in UPI-41A. It may be loaded directly from the most significant nibble of the Accumulator (MOV STS,A). These extra four STATUS bits are useful for transferring additional status information to the master. This application note uses this feature extensively.

A new instruction, EN FLAGS, allows OBF and $\overline{\text{IBF}}$ to be reflected on PORT 2 BIT 4 and PORT 2 BIT 5 respectively. This feature enables interrupt-driven data transfers when these pins are interrupt sources to the master.

By executing an EN DMA instruction PORT 2 BIT 6 becomes a DRQ (DMA Request) output and PORT 2 BIT 7 becomes $\overline{\text{DACK}}$ (DMA Acknowledge). Setting DRQ requests a DMA cycle to an external DMA controller. When the cycle is granted, the DMA controller returns DACK plus either $\overline{\text{RD}}$ (Read) or $\overline{\text{WR}}$ (Write). DACK automatically forces

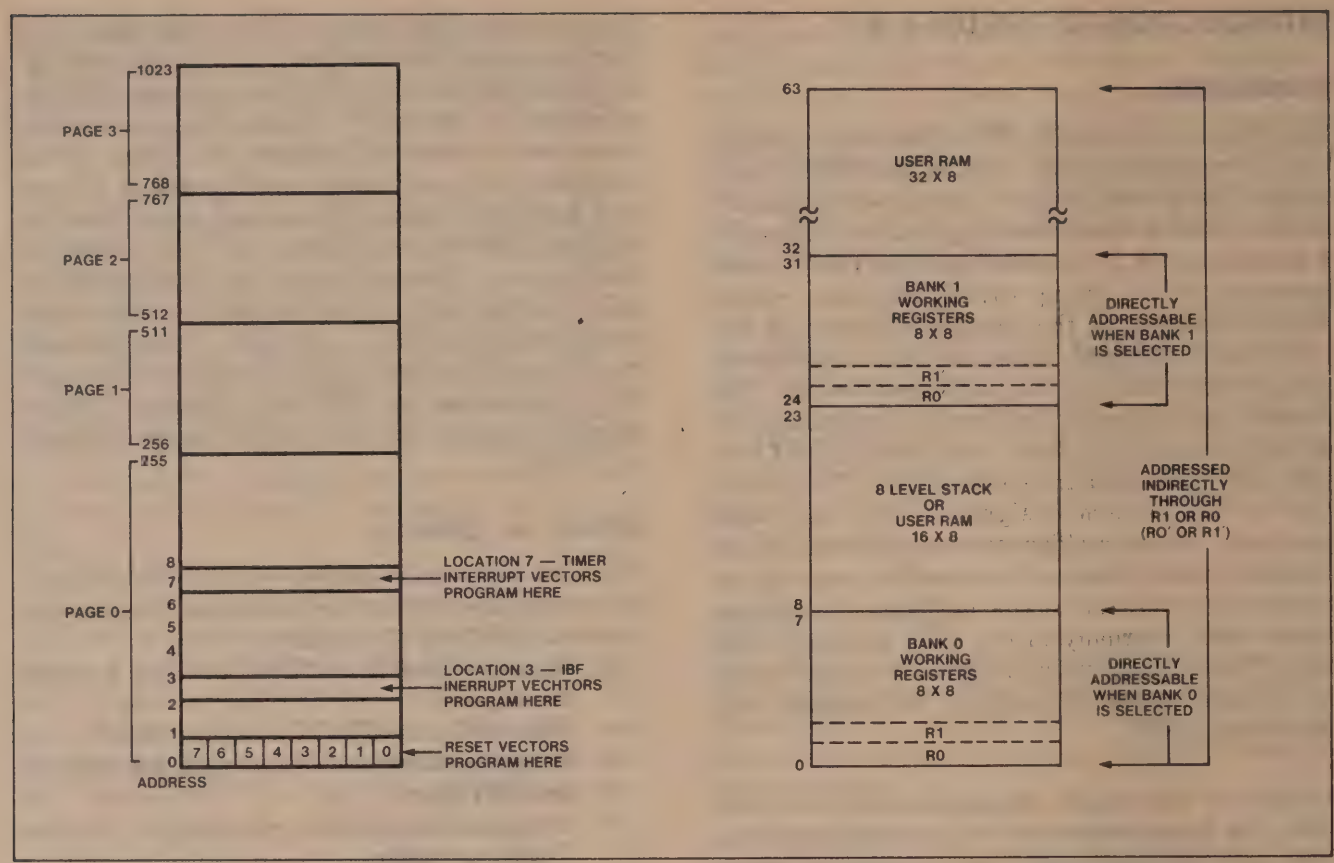


Figure 1A. Program Memory Map **Figure 1B. Data Memory Map**

$\overline{CS}$ and A_0 low internally and clears DRQ. This selects the appropriate data buffer register (DBBOUT for DACK and $\overline{RD}$, DBBIN for DACK and $\overline{WR}$) for the DMA transfer.

Like the "non-A", the UPI-41A is available in both ROM (8041A) and EPROM (8741A) Program Memory versions. This application note deals exclusively with the UPI-41A since the applications use the "A"s enhanced features.

UPI/MASTER PROTOCOL

As in most closely coupled multiprocessor systems, the various processors communicate via a shared resource. This shared resource is typically specific locations in RAM or in registers through which status and data are passed. In the case of a master processor and a UPI-41A, the shared resource is 3 separate, master-addressable, registers internal to the UPI. These registers are the status register (STATUS), the Data Bus Buffer Input register (DBBIN), and the Data Bus Output register (DBBOUT). [Data Bus Buffer direction is relative to the UPI]. To illustrate this register interface, consider the 8085A/UPI system in Figure 2.

Looking into the UPI from the 8085A, the 8085A sees only the three registers mentioned above. If the 8085A wishes to issue a command to the UPI, it does so by writing the command to the DBBIN register according to the decoding of Table 2. Data for the UPI is also passed via the DBBIN register. (The UPI differentiates commands and data by examining the A_0 pin. Just how this is done is covered shortly.) Data from the UPI for the 8085A is passed in the DBBOUT register. The 8085A may interrogate the UPI's status by reading the UPI's STATUS register. Four bits of the STATUS register act as flags and are used to handshake data and commands into and out of the UPI. The STATUS register format is shown in Figure 3.

BIT 0 is OBF (Output Buffer Full). This flag indicates to the master when the UPI has placed data in the DBBOUT register. OBF is set when the UPI writes to DBBOUT and is reset when the master reads DBBOUT. The master finds meaningful data in the DBBOUT register only when OBF is set.

The Input Buffer Full (IBF) flag is BIT 1. The UPI uses this flag as an indicator that the master has written to the DBBIN register. The master uses IBF

AFN-01536A

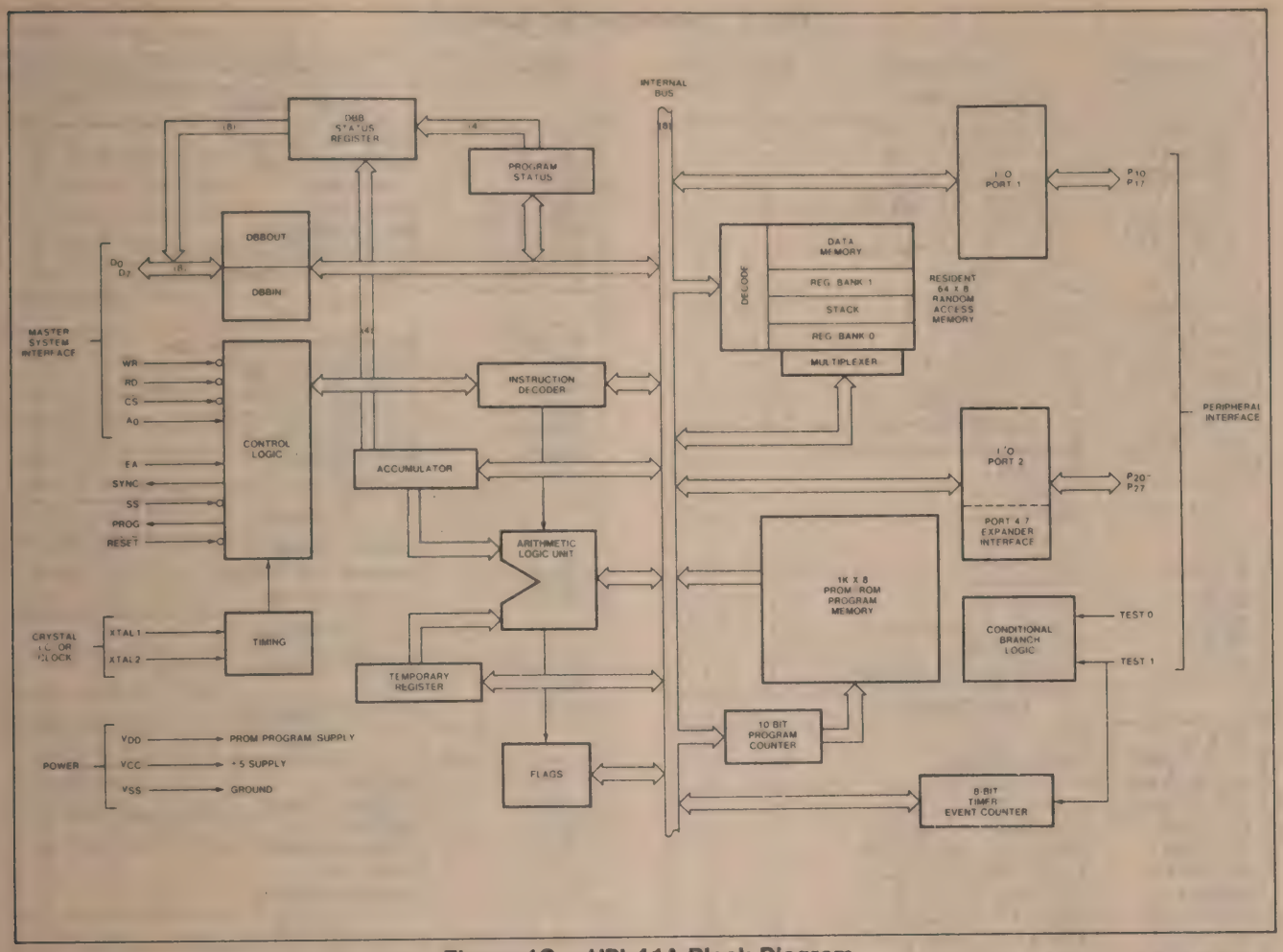


Figure 1C. UPI-41A Block Diagram

to indicate when the UPI has accepted a particular command or data byte. The master should examine IBF before outputting anything to the UPI. IBF is set when the master writes to DBBIN and is reset when the UPI reads DBBIN. The master must wait until IBF=0 before writing new data or commands to DBBIN. Conversely, the UPI must ensure IBF=1 before reading DBBIN.

The third STATUS register bit is F_0 (FLAG 0). This is a general purpose flag that the UPI can set, reset, and test. It is typically used to indicate a UPI error or busy condition to the master.

FLAG 1 (F_1) is the final dedicated STATUS bit. Like F_0 the UPI can set, reset, and test this flag. However, in addition, F_1 reflects the state of the A_0 pin whenever the master writes to the DBBIN register. The UPI uses this flag to delineate between master command and data writes to DBBIN.

The remaining four STATUS register bits are user definable. Typical uses of these bits are as status in-

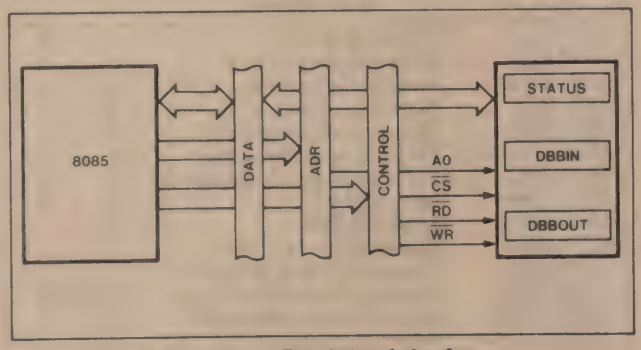


Figure 2. Register Interface

dicators for individual tasks in a multitasking UPI or as UPI generated interrupt status. These bits find a wide variety of uses in the upcoming applications.

Looking into the 8085A from the UPI, the UPI sees the two DBB registers plus the IBF, OBF, and F_1 flags. The UPI can write from its accumulator to DBBOUT or read DBBIN into the accumulator. The UPI cannot read OBF, IBF, or F_1 directly, but these flags may be tested using conditional jump

Table 1. Instruction Set Summary

Mnemonic	Description	Bytes	Cycles
Accumulator			
ADD A,R_r	Add register to A	1	1
ADD A,@R_r	Add data memory to A	1	1
ADD A,#data	Add immediate to A	2	2
ADDC A,R_r	Add register to A with carry	1	1
ADDC A @R_r	Add data memory to A with carry	1	1
ADDC A,#data	Add immed. to A with carry	2	2
ANL a,R_r	AND register to A	1	1
ANL A,@R_r	AND data memory to A	1	1
ANL A,#data	AND immediate to A	2	2
ORL A,R_r	OR register to A	1	1
ORL A@R_r	OR data memory to A	1	1
ORL A,#data	OR immediate to A	2	2
XRL A,R_r	Exclusive OR register to A	1	1
XRL A,@R_r	Exclusive OR data memory to A	1	1
XRL A,#data	Exclusive OR immediate to A	2	2
INC A	Increment A	1	1
DEC A	Decrement A	1	1
CLR A	Clear A	1	1
CPL A	Complement A	1	1
DA A	Decimal Adjust A	1	1
SWAP A	Swap digits of A	1	1
RL A	Rotate A left	1	1
RLC A	Rotate A left through carry	1	1
RR A	Rotate A right	1	1
RRC A	Rotate A right through carry	1	1
Input/Output			
IN A,P_p	Input port to A	1	2
OUTL P_p,A	Output A to port	1	2
ANL P_p,#data	AND immediate to port	2	2
ORL P_p,#data	OR immediate to port	2	2
IN A,DBB	Input DBB to A, clear IBF	1	1
OUT DBB,A	Output A to DBB, set OBF	1	1
MOV STS,A	A_4-A_7 to Bits 4-7 of Status	1	1
MOVD A,P_p	Input Expander port to A	1	2
MOVD P_p,A	Output A to Expander port	1	2
ANLD P_p,A	AND A to Expander port	1	2
ORLD P_p,A	OR A to Expander port	1	2
Data Moves			
MOV A,R_r	Move register to A	1	1
MOV A,@R_r	Move data memory to A	1	1
MOV A,#data	Move immediate to A	2	2
MOV R_r,A	Move A to register	1	1
MOV @R_r,A	Move A to data memory	1	1
MOV R_r,#data	Move immediate to register	2	2
MOV @R_r,#data	Move immediate to data memory	2	2
MOV A,PSW	Move PSW to A	1	1
MOV PSW,A	Move A to PSW	1	1
XCH A,R_r	Exchange A and register	1	1
XCH A,@R_r	Exchange A and data memory	1	1
XCHD A@R_r	Exchange digit of A and register	1	1
MOVP A,@A	Move to A from current page	1	2
MOVP3, A,@A	Move to A from page 3	1	2

Mnemonic	Description	Bytes	Cycles
Timer/Counter			
MOV A,T	Read Timer/Counter	1	1
MOV T,A	Load Timer/Counter	1	1
STRT T	Start Timer	1	1
STRT CNT	Start Counter	1	1
STOP TCNT	Stop Timer/Counter	1	1
EN TCNTI	Enable Timer/Counter Interrupt	1	1
DIS TCNTI	Disable Timer/Counter Interrupt	1	1
Control			
EN DMA	Enable DMA Handshake Lines	1	1
EN I	Enable IBF Interrupt	1	1
DIS I	Disable IBF Interrupt	1	1
EN FLAGS	Enable Master Interrupts	1	1
SEL RB0	Select register bank 0	1	1
SEL RB1	Select register bank 1	1	1
NOP	No Operation	1	1
Registers			
INC R_r	Increment register	1	1
INC @R_r	Increment data memory	1	1
DEC R_r	Decrement register	1	1
Subroutine			
CALL addr	Jump to subroutine	2	2
RET	Return	1	2
RETR	Return and restore status	1	2
Flags			
CLR C	Clear Carry	1	1
CPL C	Complement Carry	1	1
CLR F0	Clear Flag 0	1	1
CPL F0	Complement Flag 0	1	1
CLR F1	Clear F1 Flag	1	1
CPL F1	Complement F1 Flag	1	1
Branch			
JMP ADDR	Jump unconditional	2	2
JMPP @A	Jump indirect	1	2
DJNZ R,addr	Decrement register and skip	2	2
JC addr	Jump on Carry=1	2	2
JNC addr	Jump on Carry=0	2	2
JZ addr	Jump on A Zero	2	2
JNZ addr	Jump on A not Zero	2	2
JT0 addr	Jump on T0=1	2	2
JNT0 addr	Jump on T0=0	2	2
JT1 addr	Jump on T1=1	2	2
JNT1 addr	Jump on T1=0	2	2
JF0 addr	Jump on F0 Flag=1	2	2
JF1 addr	Jump on F1 Flag=1	2	2
JTF addr	Jump on Timer Flag=1,Clear Flag	2	2
JNIBF addr	Jump on IBF Flag=0	2	2
JOBF addr	Jump on OBF Flag=1	2	2
JBb addr	Jump on Accumulator Bit	2	2

Table 2. Register Decoding

CS	A0	RD	WR	REGISTER
0	0	0	1	READ DBBOUT
0	1	0	1	READ STATUS
0	0	1	0	WRITE DBBIN (DATA)
0	1	1	0	WRITE DBBIN (COMMAND)
1	X	X	X	NO ACTION

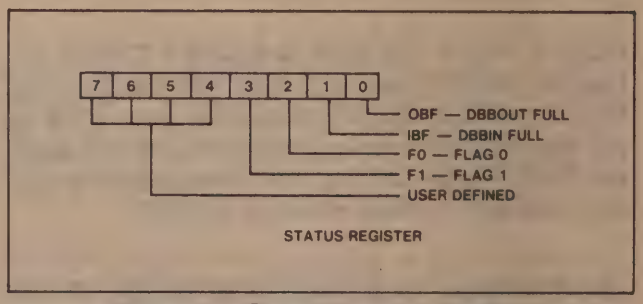


Figure 3. Status Register Format

AFN-01536A

instructions. The UPI should make sure that OBF is reset before writing new data into DBBOUT to ensure that the master has read previous DBBOUT data. IBF should also be tested before reading DBBIN since DBBIN data is valid only when IBF is set. As was mentioned earlier, the UPI uses F_1 to differentiate between command and data contents in DBBIN when IBF is set. The UPI may also write the upper 4-bits of its accumulator to the upper 4-bits of the STATUS register. These bits are thus user definable.

The UPI can test the flags at any time during its internal program execution. It essentially "polls" the STATUS register for changes. If faster response is needed to master commands and data, the UPI's internal interrupt structure can be used. If IBF interrupts are enabled, a master write to DBBIN (either command or data) sets IBF which generates an internal CALL to location 03H in program memory. At this point, working register contents can be saved using bank switching, the accumulator saved in a spare working register, and the DBBIN register read and serviced. The interrupt logic for the IBF interrupt is shown in Figure 4. A few observations concerning this logic are appropriate. Note that if the master writes to DBBIN while the UPI is still servicing the last IBF interrupt (a RETR instruction has not been executed), the IBF Interrupt Pending line is made high which causes a new CALL to 03H as soon as the first RETR is executed. No EN I (Enable Interrupt) instruction is needed to rearm the interrupt logic as is needed in an 8080 or 8085A system; the RETR performs this function. Also note that executing a DIS I to disable further IBF interrupts does not clear a pending interrupt. Only a CALL to location 03H or RESET clears a pending IBF interrupt.

Keeping in mind that the actual master/UPI protocol is dependent on the application, probably the best way to illustrate correct protocol is by example. Let's consider using the UPI as a simple parallel I/O device. (This is a trivial application but it embodies all of the important protocol considerations.) Since the UPI may be either interrupt or non-interrupt driven internally, both cases are considered.

Let's take the easiest configuration first; using the UPI PORT 1 as an 8-bit output port. From the UPI's point-of-view, this is an input-only application since all that is required is that the UPI input data from the master. Once the master writes data to the UPI, the UPI reads the DBBIN register and transfers the data to PORT 1. No testing for commands versus data is needed since the UPI "knows" it only performs one task—no commands are needed.

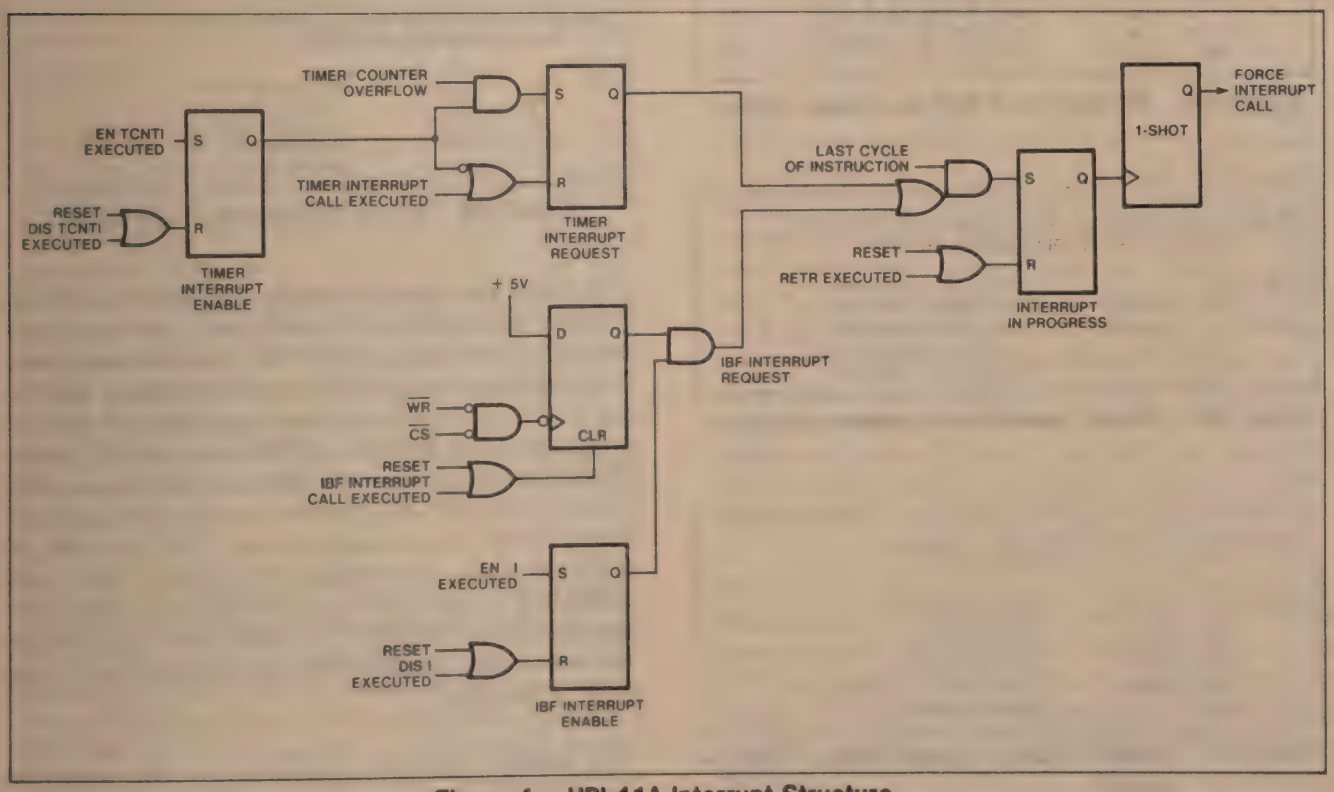


Figure 4. UPI-41A Interrupt Structure

AFN-01536A

Non-interrupt driven UPI software is shown in Figure 5A while Figure 5B shows interrupt based software. For Figure 5A, the UPI simply waits until it sees IBF go high indicating the master has written a data byte to DBBIN. The UPI then reads DBBIN, transfers it to PORT 1, and returns to waiting for the next data. For the interrupt-driven UPI, Figure 5B, once the EN I instruction is executed, the UPI simply waits for the IBF interrupt before handling the data. The UPI could handle other tasks during this waiting time. When the master writes the data to DBBIN, an IBF interrupt is generated which performs a CALL to location 03H. At this point the UPI reads DBBIN (no testing of IBF is needed since an IBF interrupt implies that IBF is set), transfers the data to PORT 1, and executes an RETR which returns program flow to the main program.

Software for the master 8085A is included in Figure 5C. The only requirement for the master to output data to the UPI is that it check the UPI to be sure the previous data had been taken before writing new data. To accomplish this the master simply reads the STATUS register looking for IBF=0 before writing the next data.

```
;
; UPI INPUT ONLY EXAMPLE—PORT 1 USED AS OUTPUT PORT
;       UPI POLLS IBF FOR DATA
;
RESET:  JNIBF  RESET      ; WAIT ON IBF FOR INPUT
        IN     A,DBB      ; INPUT THERE, SO READ IT
        OUTL   P1,A       ; TRANSFER DATA TO PORT 1
        JMP    RESET      ; GO WAIT FOR NEXT DATA
```

Figure 5A. Single Output Port Example—Polling

```
;
; UPI INPUT ONLY EXAMPLE—PORT 1 USED AS OUTPUT PORT
;       DATA INPUT IS INTERRUPT-DRIVEN ON IBF
;
RESET:  EN     I          ; ENABLE IBF INTERRUPTS
        JMP    RESET+1    ; LOOP WAITING FOR INPUT
IBFINT: IN     A,DBB      ; READ DATA FROM DBBIN
        OUTL   P1,A       ; TRANSFER DATA TO PORT 1
        RETR              ; RETURN WITH RESTORE
```

Figure 5B. Single Output Port Example—Interrupt

```
;
; 8085 SOFTWARE FOR UPI INPUT-ONLY EXAMPLE
;       DATA FOR OUTPUT IS PASSED IN REG. C
;
UPIOUT: IN     STATUS     ; READ UPI STATUS
        ANI    IBF        ; LOOK AT IBF
        JNZ    UPIOUT     ; WAIT FOR IBF=0
        MOV    A,C        ; GET DATA FROM C
        OUT    DBBIN      ; OUTPUT DATA TO DBBIN
        RET               ; DONE, RETURN
```

Figure 5C. 8085A Code for Single Output Port Example

Figure 6A illustrates the case where UPI PORT 2 is used as an 8-bit input port. This configuration is termed UPI output-only as the master does not write (input) to the UPI but simply reads either the STATUS or the DBBOUT registers. In this example only the OBF flag is used. OBF signals the master that the UPI has placed new port data in DBBOUT. The UPI loops testing OBF. When OBF is clear, the master has read the previous data and UPI then reads its input port (PORT 2) and places this data in DBBOUT. It then waits on OBF until the master reads DBBOUT before reading the input port again. When the master wishes to read the input port data, Figure 6B, it simply checks for OBF being set in the STATUS register before reading DBBOUT. While this technique illustrates proper protocol, it should be noted that it is not meant to be a good method of using the UPI as an input port since the master would never get the newest status of the port.

```
;
; UPI OUTPUT ONLY EXAMPLE—PORT 2 USED AS INPUT PORT
;       PORT DATA IS AVAILABLE IN DBBOUT
;
RESET:  JOBF   RESET      ; LOOP IF OBF=1 (DATA NOT READ)
        IN     A,P2       ; DBBOUT CLEAR, READ PORT
        OUT    DBB,A      ; TRANSFER PORT DATA TO DBBOUT
        JMP    RESET      ; WAIT FOR MASTER TO READ DATA
```

Figure 6A. Single Input Port Example

```
;
; 8085 SOFTWARE FOR UPI OUTPUT—ONLY EXAMPLE
;       INPUT DATA RETURNED IN REG. A
;
UPIIN:  IN     STATUS     ; READ UPI STATUS
        ANI    OBF        ; LOOK AT OBF
        JZ     UPIIN      ; WAIT UNTIL OBF=1
        IN     DBBOUT     ; READ DBBOUT
        RET               ; RETURN WITH DATA IN A
```

Figure 6B. 8085A Single Input Port Code

The above examples can easily be combined. Figure 7 shows UPI software to use PORT 1 as an output port simultaneously with PORT 2 as an input port. The program starts with the UPI checking IBF to see if the master has written data destined for the output port into DBBIN. If IBF is set, the UPI reads DBBIN and transfers the data to the output port (PORT 1). If IBF is not set or once the data is transferred to the output port if it was, OBF is tested. If OBF is reset (indicating the master has read DBBOUT), the input port (PORT 2) is read and transferred to DBBOUT. If OBF is set, the master has yet to read DBBOUT so the program just loops back to test IBF.

The master software is identical to the separate input/output examples; the master must test IBF

```
; UPI INPUT/OUTPUT EXAMPLE—PORT 1OUTPUT, PORT 2 INPUT

RESET:  JNIBF  OUT1      ; IF IBF=0, DO OUTPUT
        IN     A, DBB    ; IF IBF=1, READ DBBIN
        OUTL   P1, A     ; TRANSFER DATA TO PORT 1
OUT1:   JOBF   RESET     ; IF OBF=1, GO TEST IBF
        IN     A, P2     ; IF OBF=0, READ PORT 2
        OUT    DBB, A    ; TRANSFER PORT DATA TO DBBOUT
        JMP    RESET     ; GO CHECK FOR INPUT
```

Figure 7. Combination Output/Input Port Example

and OBF before writing output port data into DBBIN or before reading input port from DBBOUT respectively.

In all of the three examples above, the UPI treats information from the master solely as data. There has been no need to check if DBBIN information is a command rather than data since the applications do not require commands. But what if both PORTs 1 and 2 were used as output ports? The UPI needs to know into which port to put the data. Let's use a command to select which port.

Recall that both commands and data pass through DBBIN. The state of the A_0 pin at the time of the write to DBBIN is used to distinguish commands from data. By convention, DBBIN writes with $A_0=0$ are for data, and those with $A_0=1$ are commands. When DBBIN is written into, F_1 (FLAG 1) is set to the state of A_0. The UPI tests F_1 to determine if the information in the DBBIN register is data or command.

For the case of two output ports, let's assume that the master selects the desired port with a command prior to writing the data. (We could just use F_1 as a port select but that would not illustrate the subtle differences between commands and data). Let's define the port select commands such that BIT 1=1 if the next data is for PORT 1 (Write PORT 1=0000 0010) and BIT 2=1 if the next data is for PORT 2 (Write PORT 2=0000 0100). (The number of the set bit selects the port.) Any other bits are ignored. This assignment is completely arbitrary; we could use any command structure, but this one has the advantage of being simple.

Note that the UPI must "remember" from DBBIN write to write which port has been selected. Let's use F_0 (FLAG 0) for this purpose. If a Write PORT 1 command is received, F_0 is reset. If the command is Write PORT 2, F_0 is set. When the UPI finds data in DBBIN, F_0 is interrogated and the data is loaded into the previously selected port. The UPI software is shown in Figure 8A.

```
; UPI DUAL OUTPUT PORT EXAMPLE—BOTH PORT 1 AND 2 OUTPUTS
;       COMMAND SELECTS DESIRED PORT
;       WRITE PORT 1—0000 0010 (02H)
;       WRITE PORT 2—0000 0100 (04H)
;
;       FLAG 0 USED TO REMEMBER WHICH PORT WAS SELECTED
;       BY LAST COMMAND.
;
RESET:  JNIBF  RESET     ; WAIT FOR MASTER INPUT
        IN     A, DBB    ; READ INPUT
        JF1    CMD       ; IF F1=1, COMMAND INPUT
        JF0    PORT2     ; INPUT IS DATA, TEST F0
        OUTL   P1,A      ; F0=0, SO OUTPUT TO PORT 1
        JMP    RESET     ; WAIT FOR NEXT INPUT
PORT2:  OUTL   P2,A      ; F0=1, SO OUTPUT TO PORT 2
        JMP    RESET     ; WAIT FOR NEXT INPUT
CMD:    JB1    PT1       ; TEST COMMAND BITS (BIT 1)
        JB2    PT2       ; TEST BIT 2
        JMP    RESET     ; NEITHER BIT SET, WAIT FOR INPUT
PT1:    CLR    F0        ; PORT 1 SELECTED, CLEAR F0
        JMP    RESET     ; WAIT FOR INPUT
PT2:    CLR    F0        ; PORT 2 SELECTED, SET F0
        CPL    F0
        JMP    RESET     ; WAIT FOR INPUT
```

Figure 8A. Dual Output Port Example

Initially, the UPI simply waits until IBF is set indicating the master has written into DBBIN. Once IBF is set, DBBIN is read and F_1 is tested for a command. If $F_1=1$, the DBBIN byte is a command. Assuming a command, BIT 1 is tested to see if the command selected PORT 1. If so, F_0 is cleared and the program returns to wait for the data. If BIT 1=0, BIT 2 is tested. If BIT 2 is set, PORT 2 is selected so F_0 is set. The program then loops back waiting for the next master input. This input is the desired port data. If BIT 2 was not set, F_0 is not changed and no action is taken.

When IBF=1 is again detected, the input is again tested for command or data. Since it is necessarily data, DBBIN is read and F_0 is tested to determine which port was previously selected. The data is then output to that port, following which the program waits for the next input. Note that since F_0 still selects the previous port, the next input could be more data for that port. The port selection command could be thought of as a port select flip-flop control; once a selection is made, data may be repeatedly written to that port until the other port is selected. Master software, Figure 8B, simply must check IBF before writing either a command or data to DBBIN. Otherwise, the master software is straightforward.

For the sake of completeness, UPI software for implementing two input ports is given in Figure 9. This case is simpler than the dual output case since the UPI can assume that all writes to DBBIN are port selection commands so no command/data testing is required. Once the Port Read command is input, the selected port is read and the port data is placed in DBBOUT. Note that in this case F_0 is used as a UPI

error indicator. If the master happened to issue an invalid command (a command without either BIT 1 or 2 set), F_0 is set to notify the master that the UPI did not know how to interpret the command. F_0 is also set if the master commanded a port read before it had read DBBOUT from the previous command. The UPI simply tests OBF just prior to loading DBBOUT and if OBF=1, F_0 is set to indicate the error.

All of the above examples are, in themselves, rather trivial applications of the UPI although they could easily be incorporated as one of several tasks in a UPI handling multiple small tasks. We have covered them primarily to introduce the UPI concept and to illustrate some master/UPI protocol. Before moving on to more realistic UPI applications, let's discuss two UPI features that do not directly relate to the master/UPI protocol but greatly enhance the UPI's transfer capability.

In addition to the OBF and IBF bits in the STATUS register, these flags can also be made available directly on two port pins. These port pins can then be used as interrupt sources to the master. By executing an EN FLAGS instruction, PORT 2 pin 4 reflects the condition of OBF and PORT 2 pin 5 reflects the inverted condition of IBF ($\overline{IBF}$). These dedicated outputs can then be enabled or disabled via their respective port bit values; i.e., P_{24} reflects OBF as long as an instruction is executed which sets P_{24} (i.e. ORL P2,#10H). The same action applies to the $\overline{IBF}$ output except P_{25} is used. Thus P_{24} may serve as a DATA AVAILABLE interrupt output. Likewise for P_{25} as a READY-TO-ACCEPT-DATA interrupt. This greatly simplifies interrupt-driven master-slave data transfers.

```
;
; 8085 SOFTWARE FOR DUAL OUTPUT PORT EXAMPLE
;        THIS ROUTINE WRITES DATA IN REG. C TO PORT 1
;        (SAME ROUTINE FOR PORT 2—JUST CHANGE COMMAND)
;
PORT1:  IN    STATUS     ; READ UPI STATUS
        ANI   IBF        ; LOOK AT IBF
        JNZ   PORT1      ; WAIT UNTIL IBF=0
        MVI   A, 00000010B ; LOAD WRITE PORT1 CMD
        OUT   UPICMD     ; OUTPUT TO UPI COMMAND PORT
P1:     IN    STATUS     ; READ UPI STATUS AGAIN
        ANI   IBF        ; LOOK AT IBF
        JNZ   P1         ; WAIT UNTIL COMMAND ACCEPTED
        MOV   A, C       ; GET DATA FROM C
        OUT   DBBIN      ; OUTPUT TO DBBIN
        RET              ; DONE, RETURN
```

Figure 8B. 8085A Dual Output Port Example Code

The UPI also supports a DMA transfer interface. If an EN DMA instruction is executed, PORT 2 pin 6 becomes a DMA Request (DRQ) output and P_{27} becomes a high impedance DMA Acknowledge

```
;
; UPI DUAL INPUT PORT EXAMPLE—BOTH PORT 1 AND 2 INPUTS
;        COMMAND SELECTS WHICH PORT IS TO BE READ
;        FLAG 0 USED AS ERROR FLAG
;
RESET:  JNIBF  RESET      ; WAIT FOR INPUT
        CLR    F0         ; CLEAR ERROR FLAG
        IN     A, DBB     ; READ INPUT (COMMAND)
        JB1    PT1        ; TEST BIT 1 (PORT 1)
        JB2    PT2        ; TEST BIT 2 (PORT 2)
ERROR:  CPL    F0         ; ERROR—COMPLEMENT F0
        JMP    RESET      ; WAIT FOR INPUT
PT1:    IN     A, P1      ; READ PORT 1
        JOBF   ERROR      ; TEST OBF BEFORE LOADING DBBOUT
        OUT    DBB, A     ; LOAD PORT 1 DATA INTO DBBOUT
        JMP    RESET      ; WAIT FOR INPUT
PT2:    IN     A, P2      ; READ PORT 2
        JOBF   ERROR      ; TEST OBF BEFORE LOADING DBBOUT
        OUT    DBB, A     ; LOAD PORT 2 DATA INTO DBBOUT
        JMP    RESET      ; WAIT FOR INPUT
```

Figure 9. Dual Input Port Example

($\overline{DACK}$) input. Any instruction which would normally set P_{26} now sets DRQ. DRQ is cleared when $\overline{DACK}$ is low and either $\overline{RD}$ or $\overline{WR}$ is low. When $\overline{DACK}$ is low, $\overline{CS}$ and A0 are forced low internally which allows data bus transfers between DBBOUT or DBBIN to occur, depending upon whether $\overline{WR}$ or $\overline{RD}$ is true. Of course, the function requires the use of an external DMA controller.

Now that we have discussed the aspects of the UPI protocol and data transfer interfaces, let's move on to the actual applications.

EXAMPLE APPLICATIONS

Each of the following three sections presents the hardware and software details of a UPI application. Each application utilizes one of the protocols mentioned in the last section. The first example is a simple 8-digit LED display controller. This application requires only that the UPI perform input operations from the DBBIN; DBBOUT is not used. The reverse is true for the second application: a sensor matrix controller. The final application involves both DBBOUT and DBBIN operations: a combination serial/parallel I/O device.

The core master processor system with which these applications were developed is the iSBC 80/30 single board computer. This board provides an especially convenient UPI environment since it contains a dedicated socket specifically interfaced for the UPI-41A. The 80/30 uses the 8085A as the master processor. The I/O and peripheral complement on the 80/30 include 12 vectored priority interrupts (8 on an 8259 Programmable Interrupt Controller and 4 on the 8085A itself), an 8253 Programmable Interval Timer supplying three 16-bit programmable timers (one is dedicated as a programmable baud rate generator), a high speed serial channel provided by a 8251 Programmable USART, and 24 parallel I/O

lines implemented with an 8255A Programmable Parallel Interface. The memory complement contains 16K bytes of RAM using 2117 16K bit Dynamic RAMs and the 8202 Dynamic RAM Controller, and up to 8K bytes of ROM/EPROM with sockets compatible with 2716, 2758, or 2332 devices. The 80/30's RAM uses a dual port architecture. That is, the memory can be considered a global system resource, accessible from the on-board 8085A as well as from remote CPUs and other devices via the MULTIBUS. The 80/30 contains MULTIBUS control logic which allows up to 16 80/30s or other bus masters to share the same system bus. (More detailed information on the iSBC 80/30 and other iSBC products may be found in the latest Intel *Systems Data Catalog.*)

A block diagram of the iSBC 80/30 is shown in Figure 10. Details of the UPI interface are shown in Figure 11. This interface decodes the UPI registers in the following format:

Register	Operations
Read STATUS	IN E5H
Write DBBIN (command)	OUT E5H
Read DBBOUT (data)	IN E4H
Write DBBIN (data)	OUT E4H

8-Digit Multiplexed LED Display

The traditional method of interfacing an LED display with a microprocessor is to use a data latch along with a BDC-to-7-segment decoder for each digit of the display. Thus two ICs, seven current limiting resistors, and about 45 connections are required for each digit. These requirements are, of course, multiplied by the total number of digits desired. The obvious disadvantages of this method are high parts count and high power dissipation since each digit is "ON" continuously. Instead, a scheme of time multiplexing the display can be used to decrease both parts count and power dissipation.

Display multiplexing basically involves connecting the same segment (a, b, c, d, e, f, or g) of each digit in parallel and driving the common digit element (anode or cathode) of each digit separately. This is shown schematically in Figure 12. The various digits of the display are not all on at once; rather, only one digit at a time is energized. As each digit is energized, the appropriate segments for that digit are turned on. Each digit is enabled in this way, in sequence, at a rate fast enough to ensure that each digit appears to be "ON" continuously. This implies that the display must be "refreshed" at periodic intervals to keep the digits flicker-free. If the CPU had to handle this task, it would have to suspend normal processing, go update the display, and then return to its normal flow. This extra burden is ideally handled by a UPI. The master CPU could simply give characters to the UPI and let the UPI do the actual segment decoding, display multiplexing, and refreshing.

As an example of this technique, Figure 13 shows the UPI controlling an 8-digit LED display. All digit segments are connected in parallel and are driven through segment drivers by the UPI PORT 1. The lower 3 bits of PORT 2 are inputs to a 3-to-8 decoder which selects an individual digit through a digit driver. A fourth PORT 2 line is used as a decoder enable input. The remaining PORT 2 lines plus the TEST 0 and TEST 1 inputs are available for other tasks.

Internally, the UPI uses the counter/timer in the interval timer mode to define the interval between display refreshes. Once the timer is loaded with the desired interval and started, the UPI is free to handle other tasks. It is only when a timer overflow interrupt occurs that the UPI handles the short display multiplexing routine. The display multiplexing can be considered a background task which is entirely interrupt-driven. The amount of time spent multiplexing is such that there is ample time to handle a non-timer task in the UPI foreground. (We'll discuss this timing shortly.)

When a timer interrupt occurs, the UPI turns off all digits via the decoder enable. The next digit's segment contents are retrieved from the internal data memory and output via PORT 1 to the segment drivers. Finally, the next digit's location is placed on PORT 2 (P_{20}–P_{22}) and the decoder enabled. This displays the digit's segment information until the next interrupt. The timer is then restarted for the next interval. This process continues repeatedly for each digit in sequence.

As a prelude to discussing the UPI software, let's examine the internal data memory structure used in this application, Figure 14. This application requires only 14 of the 64 total data memory locations. The top eight locations are dedicated to the Display Map; one location for each digit. These locations contain the segment and decimal point information for each character. Just how characters are loaded into this section of memory is covered shortly. Register R_7 of Register Bank 1 is used as the temporary Accumulator store during the interrupt service routines. Register R_3 stores the digit number of the next digit to be displayed. R_2 is a temporary storage register for characters during input routine. R_0 is

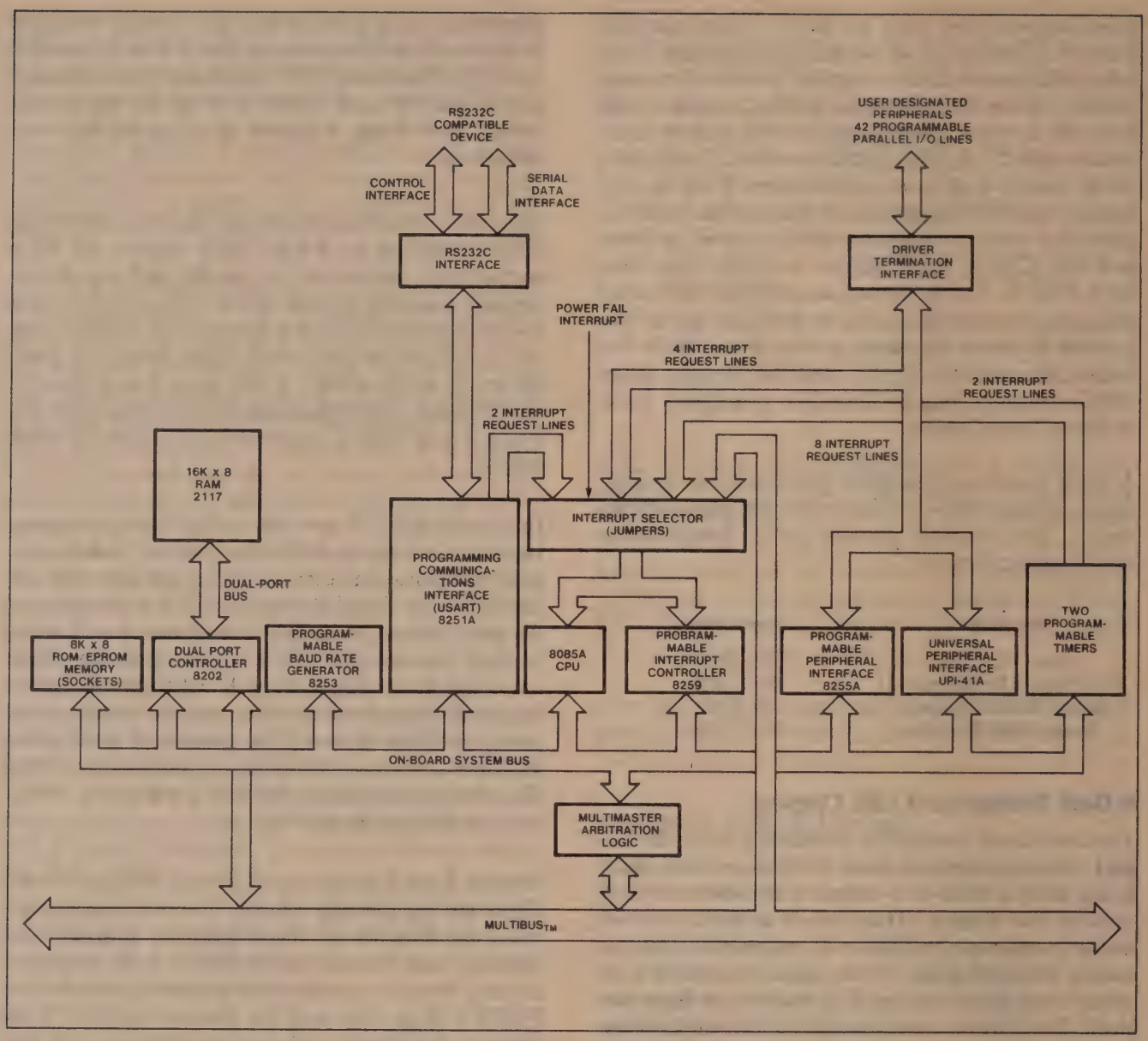


Figure 10. iSBC 80/30 Block Diagram

the offset pointer pointing to the Display Map location of the next digit. That makes 12 locations so far. The remaining two locations are the two stack locations required to store the return address plus status during the timer and input interrupt service routines. The remaining unused locations, all of Register Bank 0, 14 bytes of stack, 4 in Register Bank 1, and 24 general purpose RAM locations, are all available for use by any foreground task.

The UPI software consists of only three short routines. One, INIT, is used strictly during initialization. DISPLA is the multiplexing routine called at a timer interrupt. INPUT is the character input handler called at an IBF interrupt. The flow charts for these routines are shown in Figures 14A through 14C.

INIT initializes the UPI by simply turning off all segment and digit drivers, filling the Display Map with blank characters, loading and starting the timer, and enabling both timer and IBF interrupts. Although the flow chart shows the program looping at this point, it is here that the code for any foreground task is inserted. The only restrictions on this foreground task are that it not use I/O lines dedicated to the display and that it not require dedicated use of the timer. It could share the timer if precautions are taken to ensure that the display will still be refreshed at the required interval.

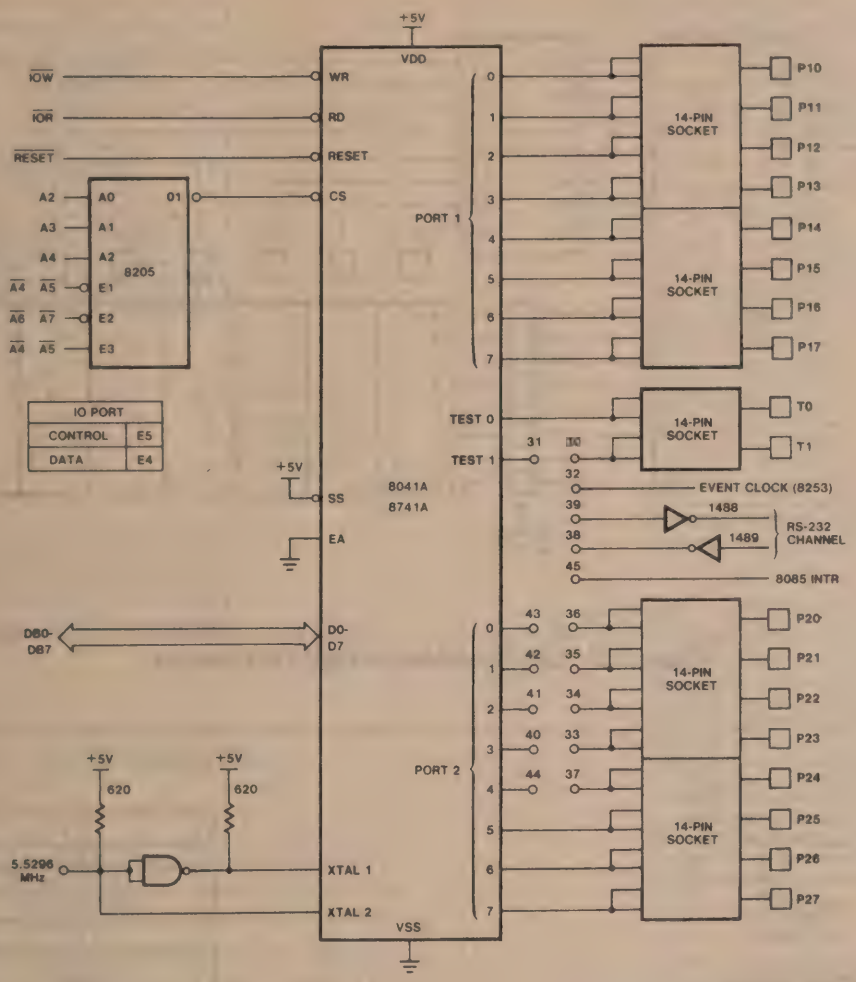


Figure 11. UPI Interface on iSBC 80/30

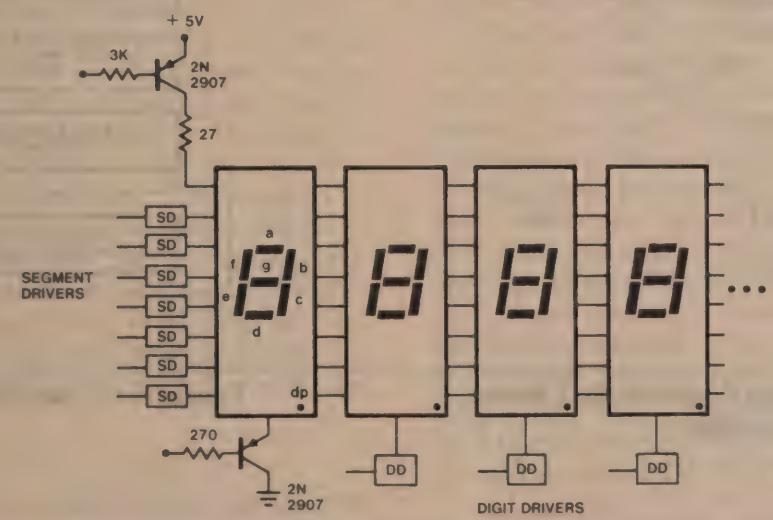


Figure 12. LED Multiplexing

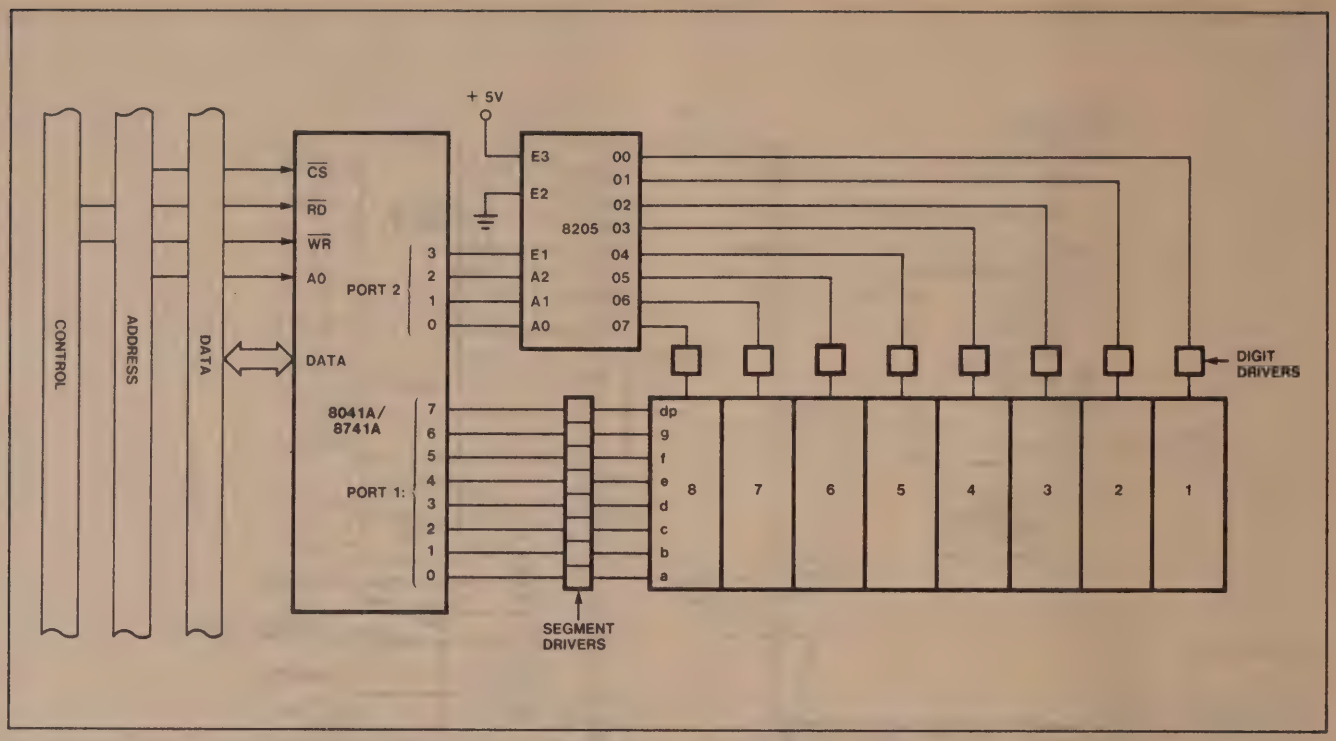


Figure 13. UPI Controlled 8-Digit LED Display

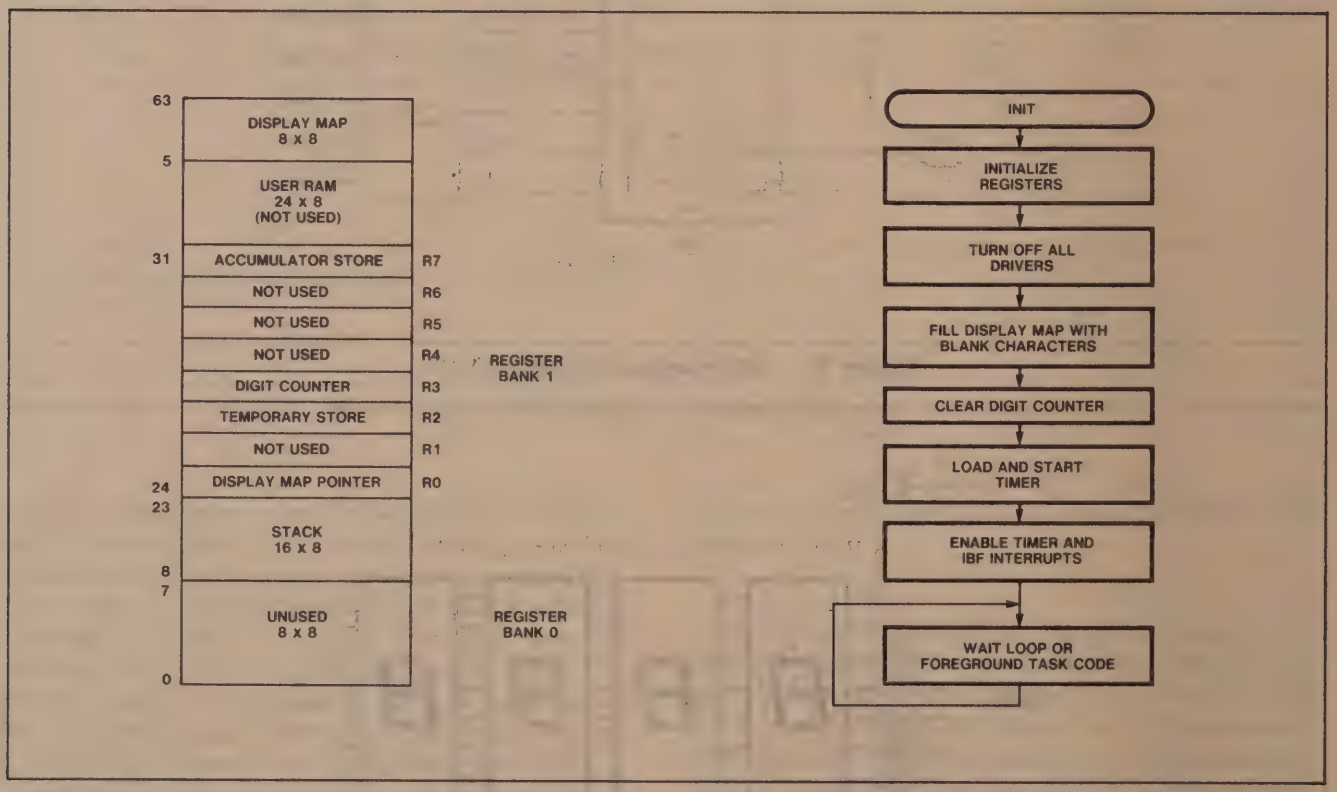


Figure 14. LED Display Controller Data Memory Allocation

Figure 14A. INIT Routine Flow

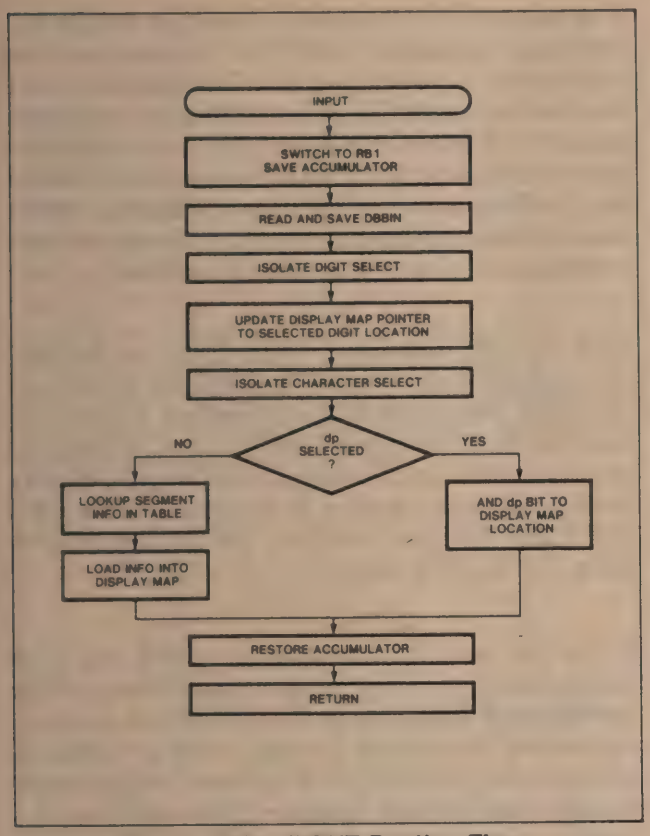


Figure 14B. INPUT Routine Flow

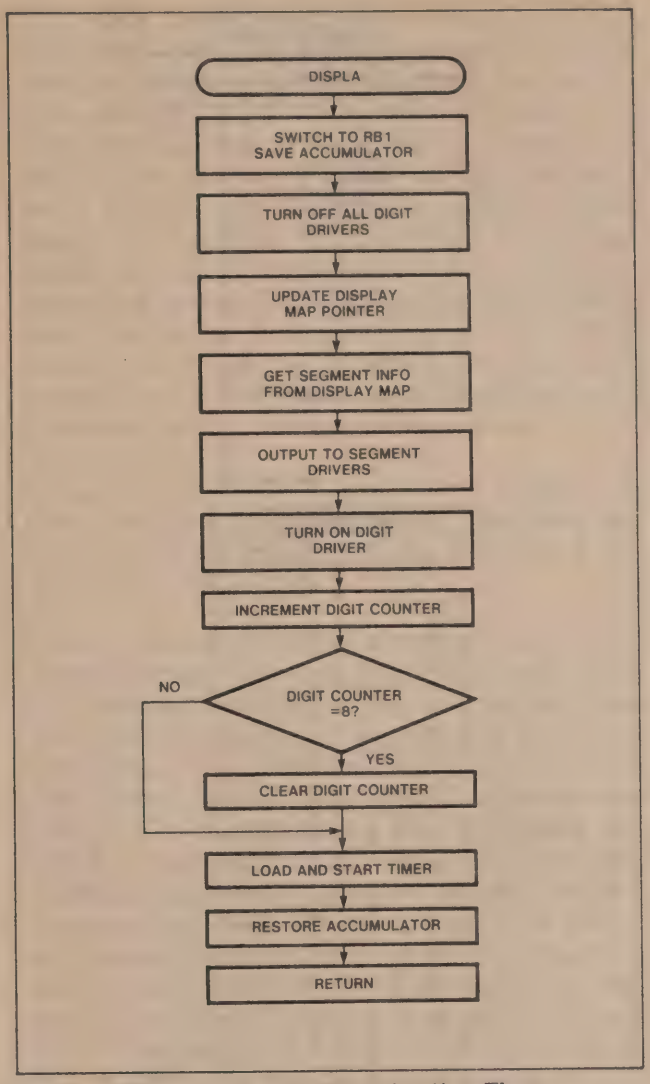


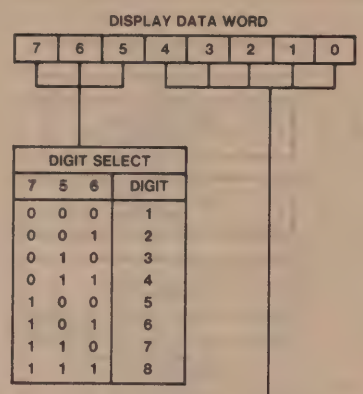
Figure 14C. DISPLA Routine Flow

The INPUT routine handles the character input. It is called when an IBF interrupt occurs. After the usual swapping of register banks and saving of the accumulator, DBBIN is read and stored in register R_2. DBBIN contains the Display Data Word. The format for this word, Figure 15, has two fields: Digit Select and Character Select. The Digit Select field selects the digit number into which the character from the Character Select field is placed. Notice that the character set is not limited strictly to numerics, some alphanumeric capability is provided. Once DBBIN is read, the offset for the selected digit is computed and placed in the Display Map Pointer R_0. Next the segment information for the selected character is found through a look-up table starting in page 3 of the program memory. This segment information is then stored at the location pointed at by the Display Map Pointer. If the Character Select field specified a decimal point, the segment corresponding to the decimal point is ANDed into the present segment information for that digit. After the accumulator is restored, execution is returned to the main program.

The DISPLA routine simply implements the multiplexing actions described earlier. It is called whenever a timer interrupt occurs. After saving pre-interrupt status by switching register banks and storing the Accumulator, all digit drivers are turned off. The Display Map Pointer is then updated using the Current Digit Register to point at that digit's segment information in the Display Map. This information is output to PORT 1; the segment drivers. The number of the current digit, R_3, is then sent to the digit select decoder and the decoder is enabled. This turns on the current digit. The digit counter is incremented and tested to see if all eight digits have been refreshed. If so, the digit counter is reset to zero. If not, nothing is done. Finally, the timer is loaded and restarted, the Accumulator is restored, and the routine returns execution to the main program. Thus DISPLA refreshes one digit each time it is CALLed by the timer interrupt. The digit remains on until the next time DISPLA is executed.

The UPI software listing is included as Appendix A1. Appendix A2 shows the 8085A test routine used

AFN-01536A



DIGIT SELECT

7	5	6	DIGIT
0	0	0	1
0	0	1	2
0	1	0	3
0	1	1	4
1	0	0	5
1	0	1	6
1	1	0	7
1	1	1	8

CHARACTER SELECT

4	3	2	1	0	CHAR
0	0	0	0	0	0
0	0	0	0	1	1
0	0	0	1	0	2
0	0	0	1	1	3
0	0	1	0	0	4
0	0	1	0	1	5
0	0	1	1	0	6
0	0	1	1	1	7
0	1	0	0	0	8
0	1	0	0	1	9
0	1	0	1	0	A
0	1	0	1	1	b
0	1	1	0	0	C
0	1	1	0	1	d
0	1	1	1	0	E
0	1	1	1	1	F
1	0	0	0	0	•
1	0	0	0	1	G
1	0	0	1	0	H
1	0	0	1	1	I
1	0	1	0	0	J
1	0	1	0	1	L
1	0	1	1	0	n
1	0	1	1	1	o
1	1	0	0	0	P
1	1	0	0	1	r
1	1	0	1	0	t
1	1	0	1	1	U
1	1	1	0	0	y
1	1	1	0	1	-
1	1	1	1	0	.
1	1	1	1	1	blank

Figure 15. LED Display Controller Display Data Word Format

to display the contents of a display buffer on the display. The 8085A software takes care of the display digit numbering. Since the application is input-only for the UPI, the only protocol required is that the master must test IBF before writing a Display Data Word into DBBIN.

On the iSBC 80/30, the UPI frequency is at 5.5296 MHz. To obtain a flicker-free display, the whole display must be refreshed at a rate of 50 Hz or greater. If we assume a 50 Hz refresh rate and an 8-digit display, this means the DISPLA routine must be CALLed 50×8 or 400 times/sec. This transfers, using the timer interval of 87 μs at 5.5296 MHz, to a timer count of 227. (Recall from the UPI-41A *User's Manual* that the timer is an "8-bit up-counter".) Hence the TIME equate of 227D in the UPI listing. Obviously, different frequency sources or display lengths would require that this equate be modified.

With the UPI running at 5.5296 MHz, the instruction cycle time is 2.713 μs. The DISPLA routine requires 28 instruction cycles, therefore, the routine executes in 76 μs. Since DISPLA is CALLed 400 times/sec, the total time spent refreshing the display during one second is then 30 ms or 3% of the total UPI time. This leaves 97.0% for any foreground tasks that could be added.

While the basic UPI software is useful just as it stands, there are several enhancements that could be incorporated depending on the application. Auto-incrementing of the digit location could be added to the input routine to alleviate the need for the master to keep track of digit numbers. This could be (optionally) either right-handed or left-handed entry a la TI or HP calculators. The character set could be easily modified by simply changing the lookup table. The display could be expanded to 16 digits at the expense of one additional PORT 2 digit select line, the replacement of the 3-to-8 decoder with a 4-to-16 decoder, and 8 more Display Map locations.

Now let's move on to a slightly more complex application that is UPI output-only—a sensor matrix controller.

Sensor Matrix Controller

Quite often a microprocessor system is called upon to read the status of a large number of simple SPST switches or sensors. This is especially true in a process or industrial control environment. Alarm systems are also good examples of systems with a large sensor population. If the number of sensors is small, it might be reasonable to dedicate a single input port pin for each sensor. However, as the number of sensors increase, this technique becomes very wasteful. A better arrangement is to configure the sensors in a matrix organization like that shown in Figure 16. This arrangement of 16 sensors requires only 4 input and 4 output lines; half the number needed if dedicated inputs were used. The line saving becomes even more substantial as the number of sensors increases.

In Figure 16, the basic operation of the matrix involves scanning individual row select lines in sequence while reading the column return lines. The state of any particular sensor can then be determined by decoding the row and column information. The typical configuration pulls up the column return lines and the selected row is held low. Deselected rows are held high. Thus a return line remains high for an open sensor on the selected row and is pulled low for a closed sensor. Diode isolation is used to prevent a phantom closure which would occur when a sensor is closed on a selected row and there are two or more closures on a deselected row. Germanium diodes are used to provide greater noise margin at the return line input.

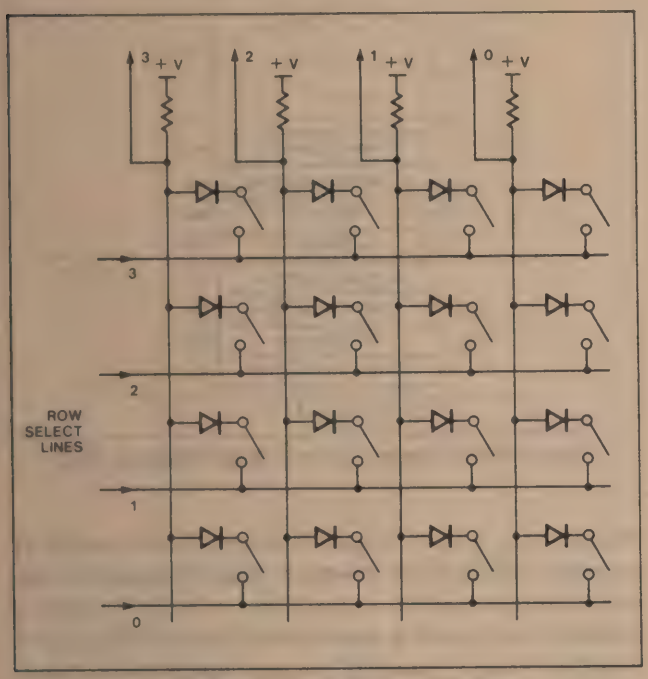


Figure 16. 4×4 Sensor Matrix

If the main processor was required to control such a matrix it would periodically have to output at the row port and then read the column return port. The processor would need to maintain in memory a map of the previous state of the matrix. A comparison of the new return information to the old information would then be made to determine whether a sensor change had occurred. Any changes would be processed as needed. A row counter and matrix map pointer also require maintenance each scan. Since in most applications sensors change very slowly compared to most processing actions, the processor probably would scan the rows only periodically with other tasks being processed between scans.

Rather than require the processor to handle the rather mundane tasks of scanning, comparing, and decoding the matrix, why not use a dedicated processor? The UPI is perfect.

Figure 17 shows a UPI configuration for controlling up to 128 sensors arranged in a 16×8 matrix. The 4-to-16 line decoder is used as the row selector to save port pins and provides the expansion to 128 sensors over the maximum of 64 sensors if the port had been used directly. It also helps increase the port drive capability. The column return lines go directly into PORT 1. Features of this design include complete matrix management. As the UPI scans the matrix it compares its present status to the previous scan. If any change is detected, the location of the change is decoded and loaded, along with the sensor's present state, into DBBOUT. This byte is called a Change Word. The Master processor has only to read one byte to determine the status and coordinate of a changed sensor. If the master had not read a previous Change Word in DBBOUT (OBF=1) before a new sensor change is detected, the new Change

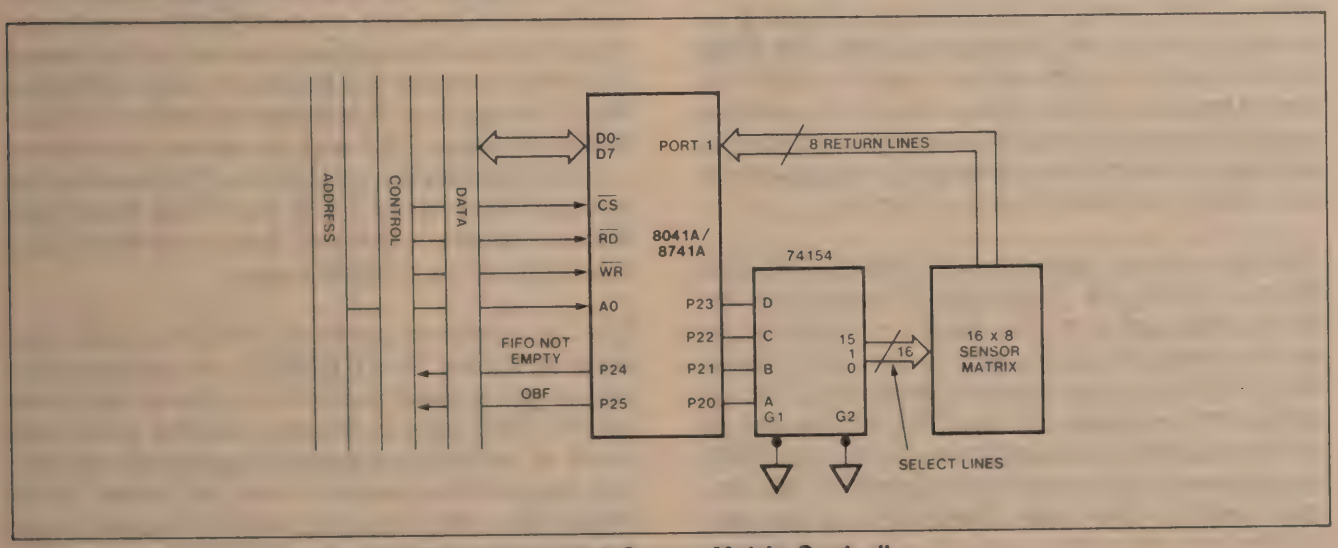


Figure 17. 128 Sensor Matrix Controller

Word is loaded into an internal FIFO. This FIFO buffers up to 40 changes before it fills. The status of the FIFO and OBF is made available to the master either by polling the UPI STATUS register, Figure 18A, or as interrupt sources on port pins P_{24} and P_{25} respectively, Figure 17. The FIFO NOT EMPTY pin and bit are true as long as there are changes not yet read in the FIFO. As long as the FIFO is not empty, the UPI monitors OBF and loads new Change Words from the FIFO into DBBOUT. Thus, the UPI provides complete FIFO management.

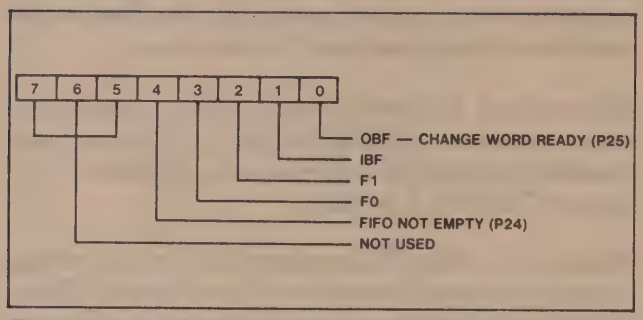


Figure 18A. Sensor Matrix Status Register Format

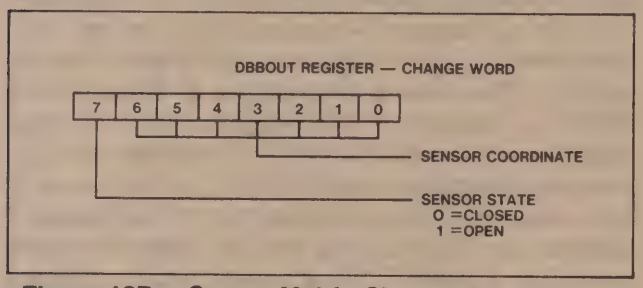


Figure 18B. Sensor Matrix Change Word Format

Internally, the matrix scanning software is programmed to run as a foreground task. This allows the timer/counter to be used by any background task although the hardware configuration leaves only 2 inputs (TEST 0 and TEST 1) plus 2 I/O port pins available. Also, to add a background task, the FIFO would have to be made smaller to accommodate the needed register and data memory space. (It would be possible however to turn the table here and make the scanning software timer/counter interrupt-driven where the timer times the scan interval.)

The data memory organization for this application is shown in Figure 19. The upper 16 bytes form the Matrix Map and store the sensor states from the previous scan; one bit for each sensor. The Change Word FIFO occupies the next 40 locations. (The top and bottom addresses of this FIFO are treated as equate variables in the program so that the FIFO size may easily be changed to accommodate the register needs of other tasks.) Register R_0 serves as a pointer into the matrix map area for comparisons and updates of the sensor status. R_1 is a general FIFO pointer. The FIFO is implemented as a circular buffer with In and Out pointer registers which are stored in R_4 and R_5 respectively. These registers are moved into FIFO pointer R_1 for actual transfers into or out of the FIFO. R_2 is the Row Select Counter. It stores the number of the row being scanned.

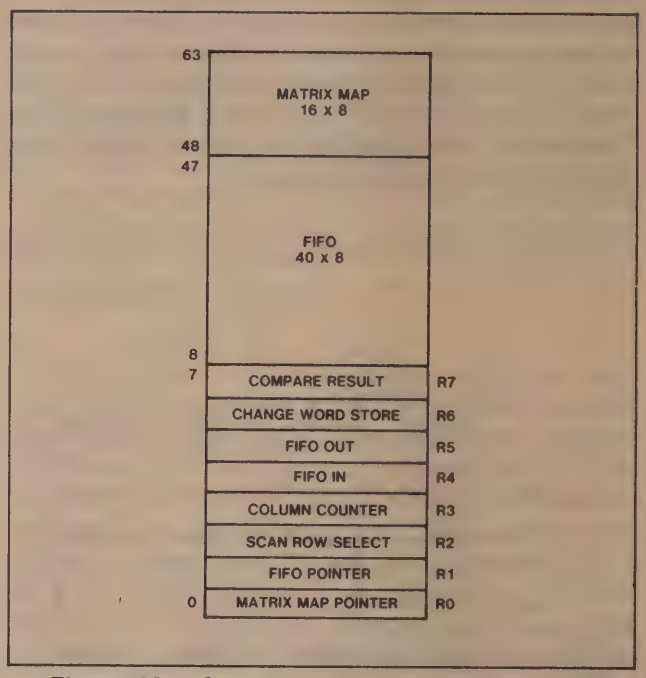


Figure 19. Sensor Matrix Data Memory Map

Register R_3 is the Column Counter. This counter is normally set to 00H; however, when a change is detected somewhere in a particular row, it is used to inspect each sensor status bit individually for a change. When a changed counter sensor bit is found, the Row Select Counter and Column Counter are combined to give the sensor's matrix coordinate. This coordinate is temporarily stored in the Change Word Store, register R_6. Register R_7 is the Compare Result. As each row is scanned, the return information is Exclusive-OR'd with the return information from the previous scan of that row. The result of this operation is stored in R_7. If R_7 is zero, there have been no changes on that row. A non-zero result indicates at least one changed sensor.

The basic program operation is shown in the flow chart of Figure 20. At RESET, the software initializes the working registers, the ports, and clears the STATUS register. To get a starting point from which to perform the sensor comparisons, the current status of the matrix is read and stored in the Matrix Map. At this point, the UPI begins looking for changed sensors starting with the first row.

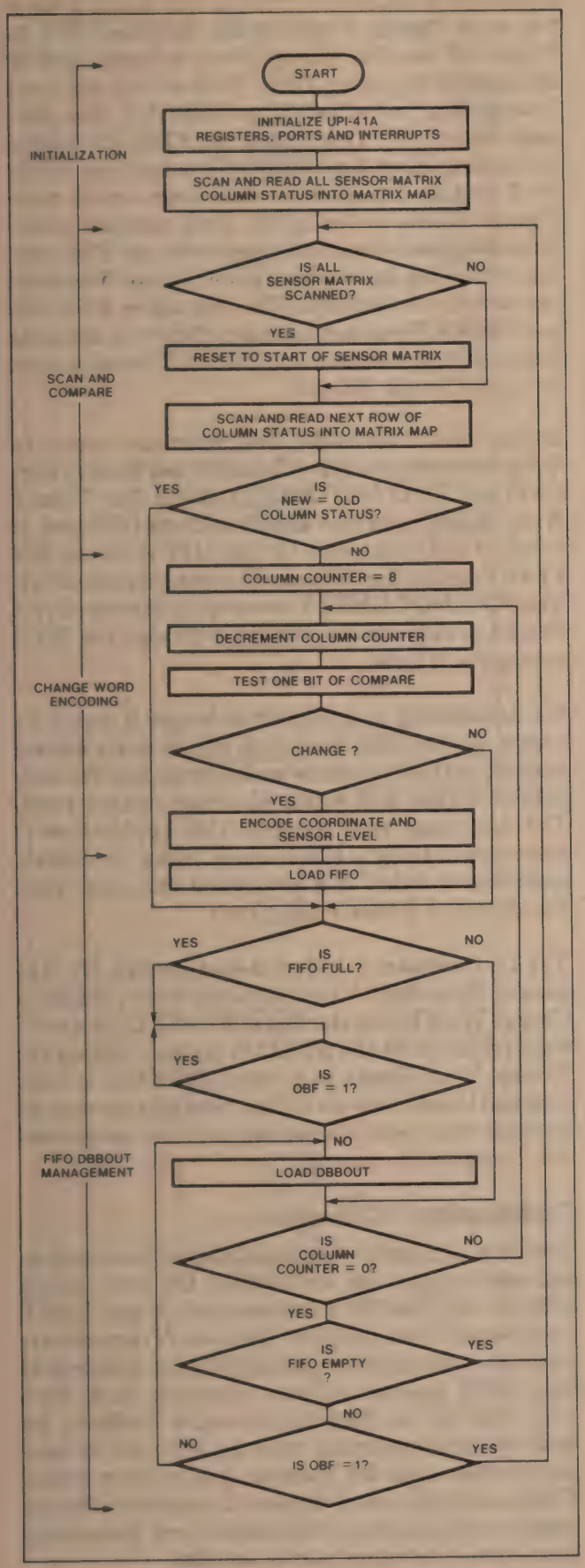


Figure 20. Sensor Matrix Controller Flow Chart

Before delving further into the flow, let's pause to describe the general format of the operation. The UPI scans the matrix one row at a time. If no changes are detected on a particular row, the UPI simply moves to the next row after checking the status of DBBOUT and the FIFO. If a change is detected, the UPI must check each bit (sensor) within the row to determine the actual sensor location. (More than one sensor on the scanned row could have changed.) Rather than test all 8 bits of the row before checking the DBBOUT and FIFO status again, the UPI performs the status check in between each of the bit tests. This ensures the fastest response to the master reading previous Change Words from DBBOUT and the FIFO.

With this general overview in mind, let's go first thru the flow chart assuming we are scanning a row where no changes have occurred. Starting at the Scan-and-Compare section, the UPI first checks if the entire matrix has been scanned. If it has, the various pointers are reset. If not, the address of the next row is placed on PORTs 20 thru 23. This selects the desired row. The state of the row is then read on PORT 1; the column return lines. This present state is compared to the previous state by retrieving the previous state from the matrix map and performing an Exclusive-OR with the present state. Since we are assuming that no change has occurred, the result is zero. No coordinate decoding is needed and the flow branches to the FIFO-DBBOUT Management section.

The FIFO-DBBOUT Management section simply maintains the FIFO and loads DBBOUT whenever Change Words are present in the FIFO and DBBOUT is clear (OBF=0). The section first tests if the FIFO is full. (If we assume our "no-change" row is the first row scanned, the FIFO obviously would not be full.) If it is, the UPI waits until OBF=0, at which point the next Change Word is retrieved from the FIFO and placed in DBBOUT. This "unfills" the FIFO making room for more Change Words. At this point, the Column Counter, R_3, is checked. For rows with no changes, the Column Counter is always zero so the test simply falls through. (We cover the case for changes shortly.) Now the FIFO is tested for being empty. If it is, there is no sense in any further tests so the flow simply goes back up to scan the next row. If the FIFO is not empty, DBBOUT is tested again through OBF. If a Change Word is in DBBOUT waiting for the master to read it, nothing can be done and the flow likewise branches up for the next row. However, if the DBBOUT is free and remembering that the previous test showed that the FIFO was not empty, DBBOUT is loaded with the next Change Word and the last two conditional tests repeat.

Now let's assume the next row contains several changed sensors. Like before, the row is selected, the return lines read, and the sensor status compared to the previous scan. Since changes have occurred, the Exclusive-OR result is now non-zero. Any 1's in the result reflect the positions of the changed sensors. This non-zero result is stored in the Compare Result register, R_7. At this point, the Column Counter is preset to 8. To determine the changed sensors' locations, the Compare Result register is shifted bit-by-bit to the left while decrementing the Column Counter. After each shift, BIT 7 of the result is tested. If it is a one, a changed sensor has been found. The Column Counter then reflected the sensor's matrix column position while the Scan Row Select register holds it row position. These registers are then combined in R_6, the Change Word Store, to form the sensor's matrix coordinate section of the Change Word. The 8th bit of the Change Word Store is coded with the sensor's present state (Figure 18). This byte forms the complete Change Word. It is loaded into the next available FIFO position. If BIT 7 of the Compare Result had been zero, that particular sensor had not changed and the coordinate decoding is not performed.

In between each shift, test, and coordinate encode (if necessary), the FIFO-DBBOUT Management is performed. It is the Column Counter test within this section that routes the flow back up to the Change Word Encoding section if the entire Compare Result (row) has not been shifted and tested.

The FIFO is implemented as a circular buffer with IN and OUT pointers (R_4 and R_5 respectively). The operations of the FIFO is best understood using an example, Figure 21. This series of figures show how the FIFO, DBBOUT, and OBF interact as changes are detected and Change Words are read by the master. The letters correspond to sequential Change Words being loaded into the FIFO. Note that the figures show only a 4×8 FIFO however, the principles are the same in the 40×8 FIFO.

Figure 21A shows the condition where no Change Words have been loaded into the FIFO or DBBOUT. In Figure 21B a change, "A", has been detected, decoded, and loaded into the FIFO at the location equal to the value of the FIFO-IN pointer. The FIFO-OUT pointer is reset to the bottom of the FIFO since it had reached the FIFO top. Now that a Change Word is in the FIFO, OBF is checked to see if DBBOUT is empty. Because OBF=0, DBBOUT is empty and the Change Word is loaded from the FIFO location pointed at by the FIFO-OUT pointer. This is shown in Figure 21C. Loading DBBOUT automatically sets OBF. OBF remains set until the master reads DBBOUT. Figures 21D and 21E show two more Change Words loaded into the FIFO. In Figure 21F the first Change Word is finally read by the master resetting OBF. This allows the next Change Word to be loaded into DBBOUT. Note that each time the FIFO is loaded, the FIFO-IN pointer increments. Each time DBBOUT is read the FIFO-OUT pointer increments unless there are no more Change Words in the FIFO. Both pointers wrap-around to the bottom once they reach the FIFO top. The remaining figures show more Change Words being loaded into the FIFO. When the entire FIFO fills and DBBOUT can not be loaded (OBF=1), scanning stops until the master reads DBBOUT making room for more Change Words.

As was mentioned earlier, two interrupt outputs to the master are available: Change Word Ready (P_{25}, OBF) and FIFO NOT EMPTY (P_{24}). The Change Word Ready interrupt simply reflects OBF and is handled automatically by the UPI since an EN FLAGS instruction is executed during initialization. The FIFO NOT EMPTY interrupt is generated and cleared as appropriate, each pass through the FIFO management code.

No debouncing is provided although it could be added. Rather, the scan time is left as an equate variable so that it could be varied to account for both debounce time and expected sensor change rates. The minimum scan time for this application is 2msec when using a 6MHz clock. Since the matrix controller is coded as a foreground task, scan time simply uses a software delay loop.

The UPI software is included as Appendix B1. Appendix B2 is 8085A test software which builds a Change Word buffer starting at BUFSRT. This software simply polls the STATUS register looking for Change Word Ready to go true. DBBOUT is then read and loaded into the buffer. Now let's move on to an application which combines both the foreground and background concepts.

Combination I/O Device

The final UPI application was designed especially to add additional serial and parallel I/O ports to the iSBC 80/30. This UPI simulates a full-duplex UART (Universal Asynchronous Receiver/Transmitter) combined with an 8-bit parallel I/O port. Features of the UART include: software selectable baud rates (110, 300, 600, or 1200 baud), double buffering for both the transmitter and receiver, and receiver testing for false start bit, framing, and overrun errors. For parallel I/O, one 8-bit port is programmable for either input or output. The output port is statically latched and the input port is sampled.

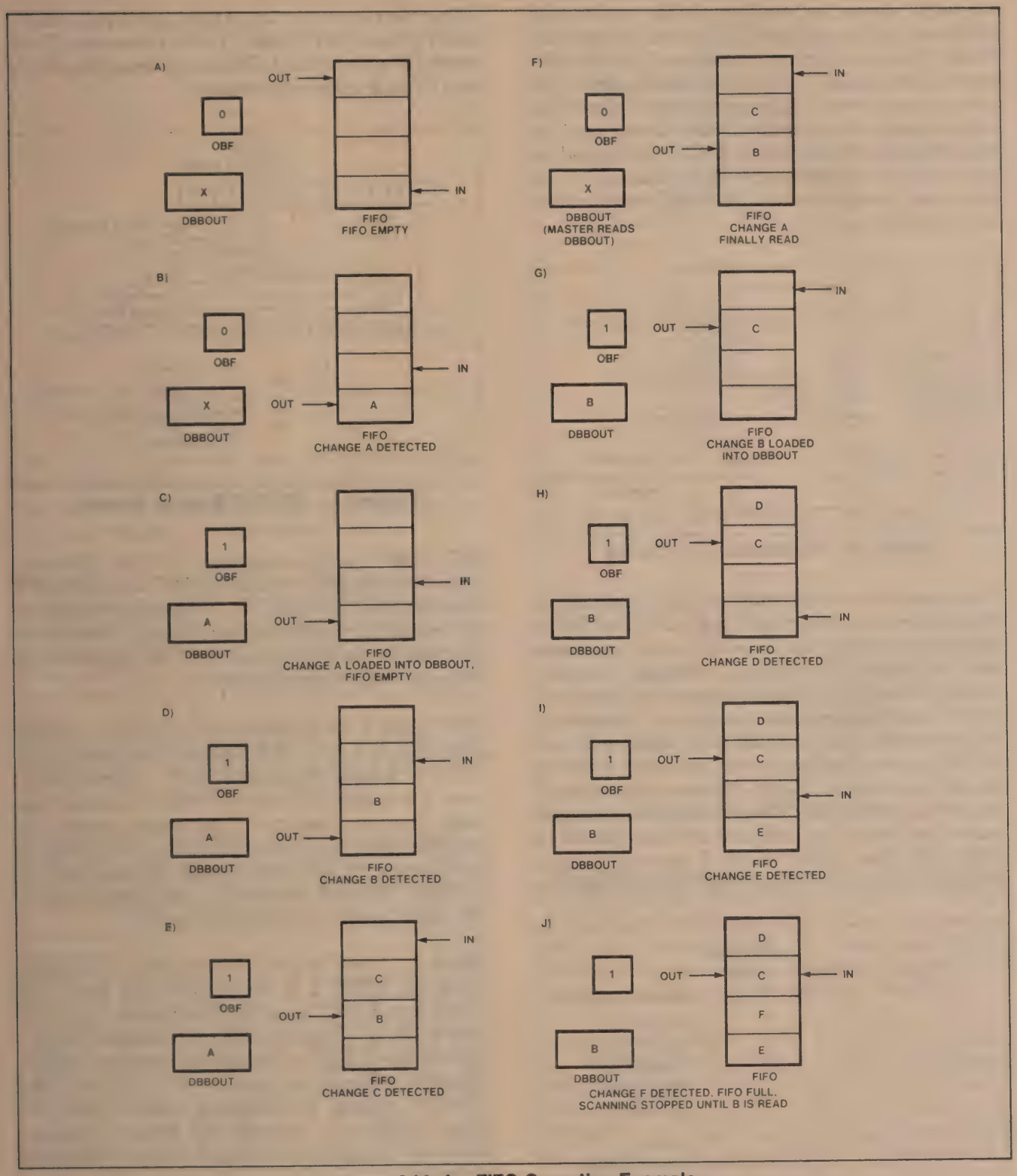


Figure 21A-J. FIFO Operation Example

Figure 22 shows the interface of this combination I/O device to the dedicated UPI socket on the iSBC 80/30. The only external requirement is a 76.8 kHz source which serves as the baud rate standard. The internal baud rates are generated as multiples of this external clock. This clock is obtained from one of the 8253 counters. Otherwise, an RS-232 driver and receiver already available for UPI use in serial I/O applications. Sockets are also provided for termination of the parallel port.

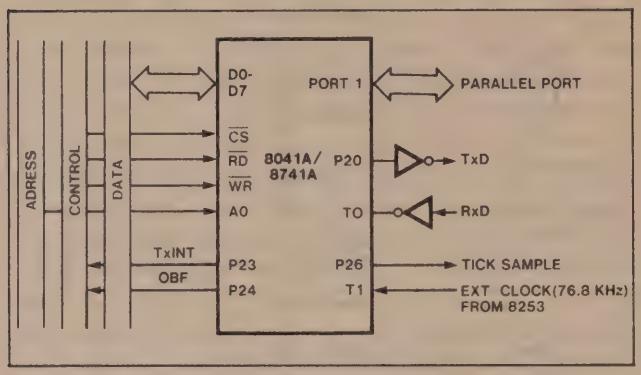


Figure 22. Combination I/O Device

There are three commands for this application. Their format is shown in Figure 23. The CONFIGURE command specifies the serial baud rate and the parallel I/O direction. Normally this command is issued once during system initialization. The I/O command causes a parallel I/O operation to be performed. If the parallel port direction is out, the UPI expects the data byte immediately following an I/O command to be data for the output port. If the port is in the input direction, an I/O command causes the port to be read and the data placed in DBBOUT. The RESET ERROR command resets the serial receiver error bits in the STATUS register.

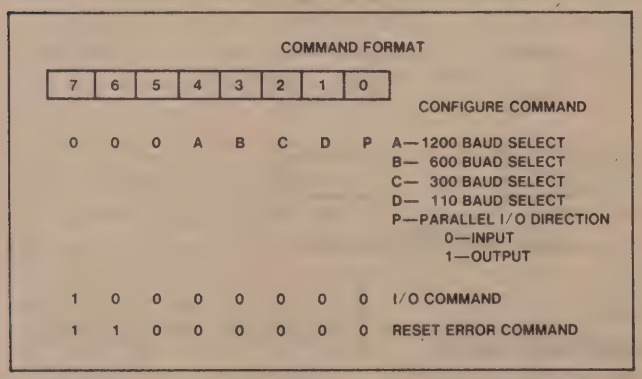


Figure 23. Combination I/O Command Format

The STATUS register format is shown in Figure 24. Looking at each bit, BIT 0 (OBF) is the DATA AVAILABLE flag. It is set whenever the UPI places data into DBBOUT. Since the data may come from either the receiver or the parallel input port, the F_0 and F_1 flags (BITs 2 and 3) code the source. Thus, when the master finds OBF set, it must decode F_0 and F_1 to determine the source.

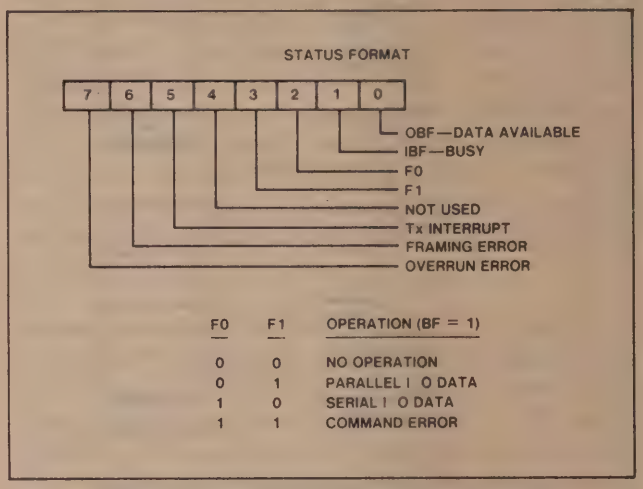


Figure 24. STATUS Register Format

BIT 1 (IBF) functions as a busy bit. When IBF is set, no writes to DBBIN are allowed. BIT 5 is the TxINT (Transmitter Interrupt) bit. It is asserted whenever the transmitter buffer register is empty. The master uses this bit to determine when the transmitter is ready to accept a data character.

BITS 6 and 7 are receiver error flags. The framing error flag, BIT 6, is set whenever a character is received with an invalid stop bit. BIT 7, overrun error, is set if a character is received before the master has read a previous character. If an overrun occurs, the previous character is overwritten and lost. Once an error occurs, the error flag remains set until reset by a RESET ERROR command. A set error flag does not inhibit receiver operation however.

Figure 25 shows the port pin definition for this application. PORT 1 is the parallel I/O port. The UART uses PORT 2 and the Test inputs. P_{20} is the transmitter data out pin. It is set for a mark and reset for a space. P_{23} is a transmitter interrupt output. This pin has the same timing as the TxINT bit in the STATUS register. It is normally used in interrupt-driven systems to interrupt the master processor when the transmitter is ready to accept a new data character.

The OBF flag is brought out on P_{24} as a master interrupt when data is available in DBBOUT. P_{26} is a diagnostic pin which pulses at four times the selected baud rate. (More about this pin later.) The receiver data input uses the TEST 0 input. One of the PORT 2 pins could have been used, however, the

PORT PIN DEFINITION

PORT	BIT	FUNCTION
1	0-7	PARALLEL I/O
2	0	Tx Data
	1	NOT USED
	2	NOT USED
	3	Tx INTERRUPT
	4	OBF INTERRUPT
	5	NOT USED
	6	NOT USED (TICK SAMPLE)
	7	NOT USED
T0		Rx DATA
T1		EXTERNAL CLOCK (76.8 kHz)

Figure 25. Combination I/O Port Definition

software can test the TEST 0 in one instruction without first reading a port.

The TEST 1 input is the baud rate external source. The UART divides this input to determine the timing needed for the selected baud rate. The input is a non-synchronous 76.8 kHz source.

Internally, when the CONFIGURE command is received and the selected baud rate is determined, the internal timer/counter is loaded with a baud rate constant and started in the event counter mode. Timer/counter interrupts are then enabled. The baud rate constant is selected to provide a counter interrupt at four times the desired baud rate. At each interrupt, both the transmitter and receiver are handled. Between interrupts, any new commands and data are recognized and executed.

As a prelude to discussing the flow charts, Figure 26 shows the register definition. Register Bank 0 serves the UART receiver and parallel I/O while Register Bank 1 handles the UART transmitter and commands. Looking at RB0 first, R_3 is the receiver status register, RxSTS. Reflected in the bits of this register is the current receiver status in sequential order. Figure 27 shows this bit definition. BIT 0 is the Rx flag. It is set whenever a possible start bit is received. BIT 1 signifies that the start bit is good and character construction should begin with the next received bit. BIT 1 is the Good Start flag. BIT 2 is the Byte Finished flag. When all data bits of a character are received, this flag is set. When all the bits, data and stop bits are received, the assembled character is loaded into the holding register (R_4 in Figure 27) BIT 3, the Data Ready flag, is set. The foreground routine which looks for commands and data continuously, looks at this bit to determine when the receiver has received a character. BITS 4 and 5 signify any error conditions for a particular character.

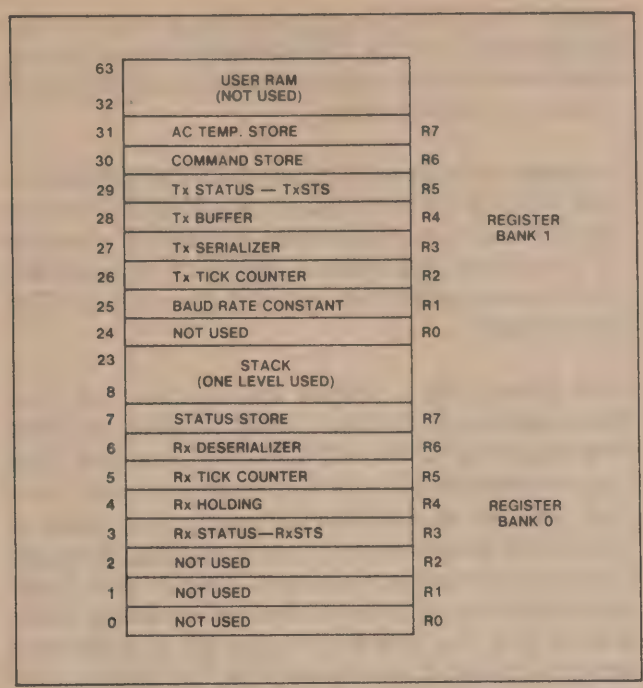


Figure 26. Combination I/O Register Map

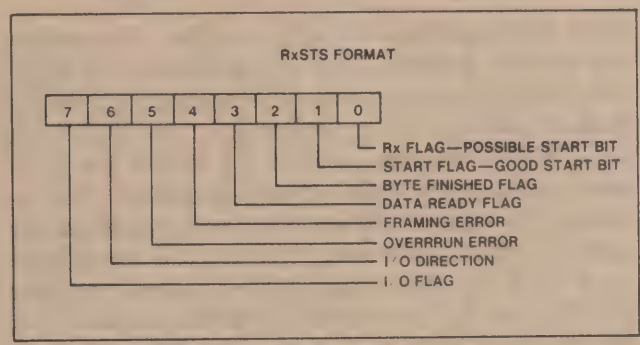


Figure 27. RxSTS Register

The parallel I/O port software uses BITS 6 and 7. BIT 6 codes the I/O direction specified by the last CONFIGURE command. BIT 7 is set whenever an I/O command is received. The foreground routine tests this bit to determine when an I/O operation has been requested by the master.

As was mentioned, R_4 is the receiver holding register. Assembled characters are held in this register until the foreground routine finds DBBOUT free, at which time the data is transferred from R_4 to DBBOUT. R_5 is the receiver tick counter. Recall that counter interrupts occur at four times the baud rate. Therefore, once a start bit is found, the receiver only needs to look at the data every four interrrupts or tick counts. R_5 holds the current tick count.

R_6 is the receiver de-serializing register. Data characters are assembled in this register. R_6 is preset to 80H when a good start bit is received. As each bit is

sampled every four timer ticks, they are rotated into the leftmost bit of R_6. The software knows the character assembly is complete when the original preset bit rotates into the carry.

An image of the upper 4 bits of the STATUS register is stored in R_7. These bits are the TxINT, Framing and Overrun bits. This image is needed since the UPI may load the upper 4 STATUS register bits from its accumulator; however, it cannot read STATUS directly.

In Register Bank 1 (Figure 26), R1 holds the baud rate constant which is found from decoding the baud rate select bits of the CONFIGURE command. The counter is reloaded with this constant every timer tick. Like the receiver, the transmitter only needs to update the transmitter output every four ticks. R_2 holds the transmitter tick count. The value of R_2 determines which portion of the data is being transmitted; start bit, data bits, or stop bit. The transmit serializer is R_3. R_3 holds the data character as each character bit is transmitted.

R_4 is the transmitter holding register. It provides the double buffering for the transmitter. While transmitting one character, it is possible to load the next character into R_4 via DBBIN. The TxINT bit in STATUS and pin on PORT 2 reflect the "fullness" of R_4. If the holding register is empty, the interrupt bit and pin are set. They are reset when the master writes a new data byte for the transmitter into DBBIN. The transmitter status register (TxSTS) is R_5. Like RxSTS,TxSTS contains flag bits which indicate the current state of the transmitter. This flag bit format is shown in Figure 28.

TxSTS BIT 0 is the Tx flag. It is set whenever the transmitter is transmitting a character. It is set from the beginning of the start bit until the end of the stop bit. BIT 1 is the Tx request flag. This bit is set by the foreground routine when it transfers a new character from DBBIN to the Tx holding register, R_4. The transmitter software uses this flag to tell if new data is available. It is reset when the transmitter transfers the character from the holding register to the serializer.

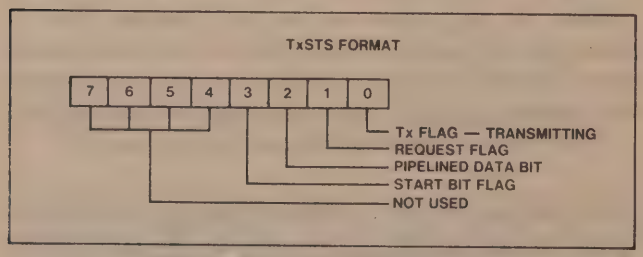


Figure 28. TxSTS Register

BIT 2 is the pipelined Tx data bit. The transmitter uses a pipelining technique which sets up the next output level in BIT 2 after processing the current timer tick. The output level is always changed at the same point after a timer tick interrupt. This technique ensures that no bit timing distortion results from different length processing paths through the receiver and transmitter routines.

BIT 3 of TxSTS is the Start Bit flag. It is set by the transmitter when the start bit space is set up in the pipelined data bit. This allows the transmitter to differentiate between the start bit and the data bits on following timer ticks.

The flow charts for this application are shown in Figures 29A–F. At reset, the INIT routine is executed which initializes the registers and port pins. After initialization, IBF and OBF are tested in MNLOOP. These flags are tested continually in this loop. If IBF is set, F_1 is tested for command or data and execution is transferred to the appropriate routine (CMD or DATA). If IBF=0, OBF is checked. If OBF=0 (DBBOUT is free), the Rx data ready and I/O flags in RxSTS are tested. If Rx data ready is set, the received data is retrieved from the Rx holding register and transferred to DBBOUT. Any error flags associated with that data are also transferred to STATUS. If the I/O flag is set and the I/O direction is input, PORT 1 is read and the data transferred to DBBOUT. In either case, F_0 and F_1 are set to indicate the data source.

If IBF is set by a command write to DBBIN, CMD reads the command and decodes the desired operation. If an I/O operation is specified, the I/O flag is set to indicate to the MNLOOP and DATA routines that an I/O operation is to be performed. If the command is a CONFIGURE command, the constant for the selected baud rate is loaded into both Baud Rate Constant register and the timer/counter. The timer/counter is started in the event counter mode and timer/counter interrupts are enabled. In addition, the I/O port is initialized to all 1's if the I/O direction bit specifies an input port. If the command is a RESET ERROR command, the two error flags in STATUS are cleared.

If the IBF flag is set by a data write, the DATA routine reads DBBIN and places the data in the appropriate place. If the I/O flag is set, the data is for the output port so the port is loaded. If the I/O flag is reset, the data is for the UART transmitter. Data for the transmitter resets the TxINT bit and pin plus sets the Tx request flag in TxSTS. The data is transferred to the Tx holding register, R_4.

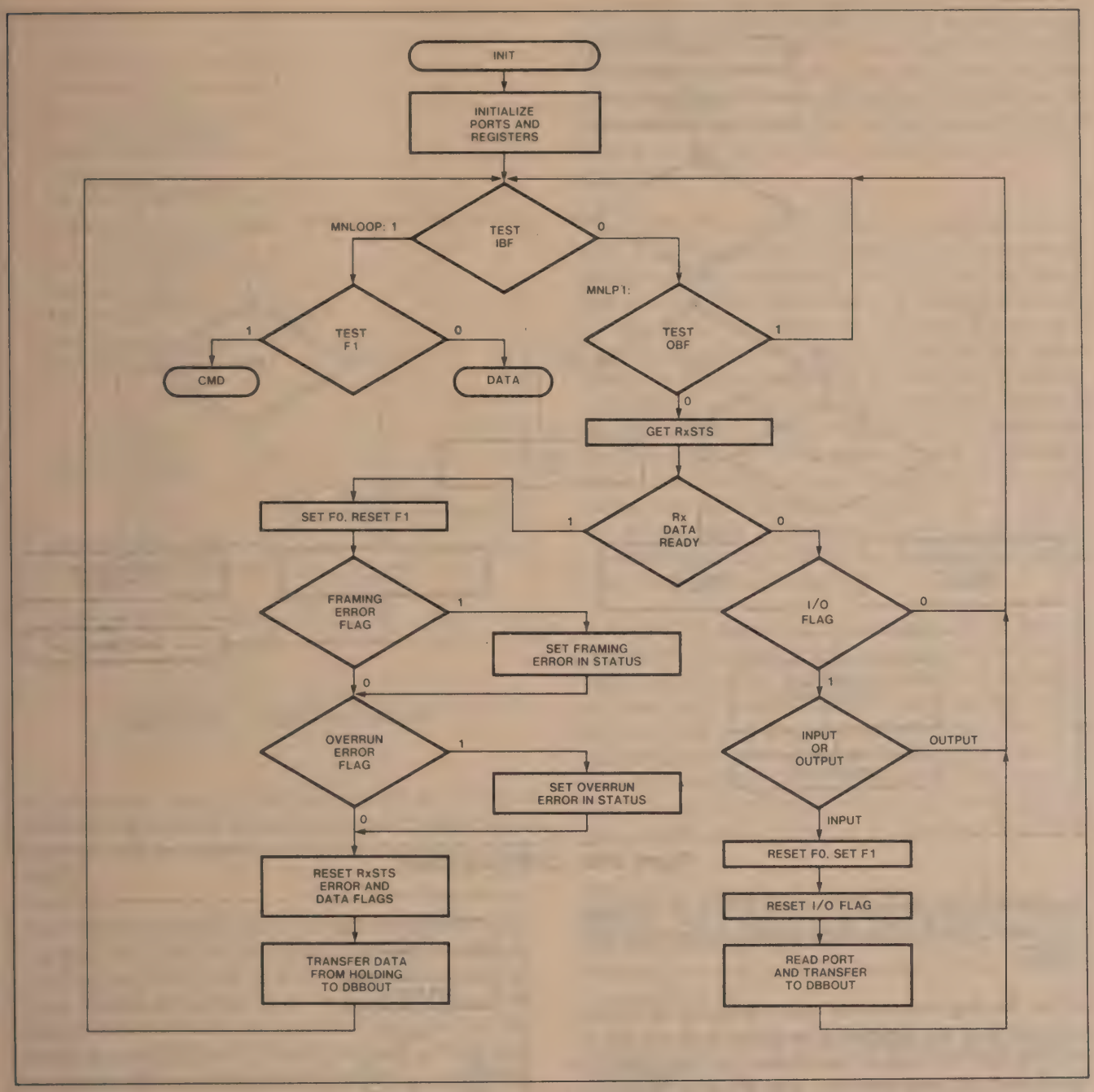


Figure 29A. INIT Flow Chart

Once a CONFIGURE command is received and the counter started, timer/counter interrupts start occurring at four times the selected baud rate. These interrupts cause a vector to the TIMINT routine, Figure 29D. A 76.8 kHz counter input provides a 13.02 μs counter resolution. Since it requires several UPI instruction cycles to reload the counter, the counter is set to two counts less than the desired baud rate and the counter is reloaded in TIMINT synchronous with the second low-going transition after the interrupt. Once the counter is reloaded, an output port (P_{26}) is toggled to give an external indication of internal counter interval. This is a helpful diagnostic feature. After the tick sample output, the pipelined transmitter data in TxSTS is output to the TxD pin. Although this occurs every timer tick, the pipelined data is changed only every fourth tick.

The receiver is now handled, Figure 29E. The Rx flag in RxSTS is examined to see if the receiver is currently in the process of receiving a character. If it is not, the RxD input is tested for a space condition which might indicate a possible start bit. If the input is a mark, no start bit is possible and execution

AFN-01536A

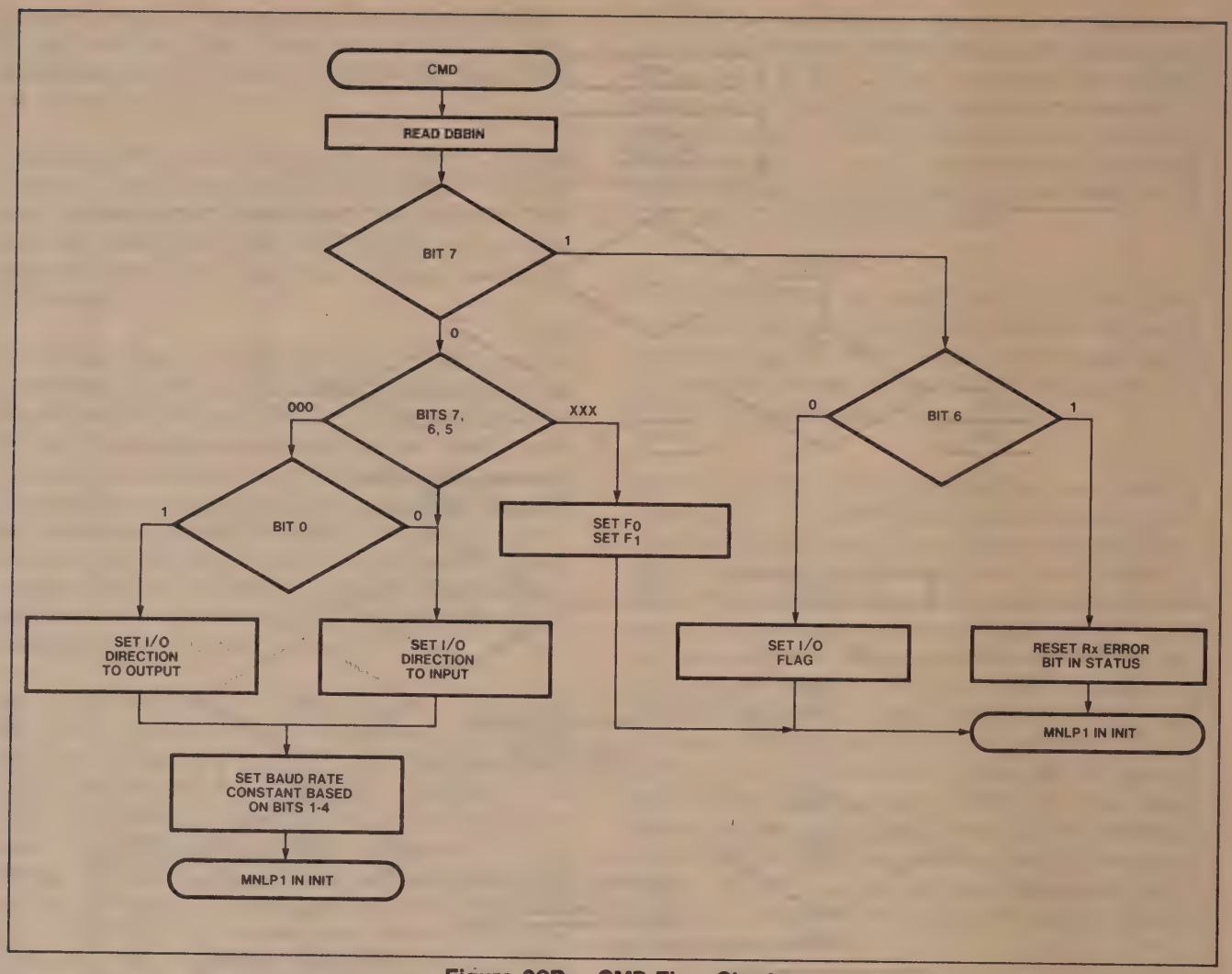


Figure 29B. CMD Flow Chart

branches to the transmitter flow, XMIT. If the input is a space, the Rx flag is set before proceeding with XMIT.

If the Rx flag is found set when entering RCV, the receiver is in the process of receiving a character. If so, the start bit flag is then tested to determine if a good start bit was received. The Rx tick counter is initialized to 4 and the Rx deserializer is set to 80H. A mark indicates a bad start bit; the Rx flag is reset to abort the reception.

If the start bit flag is set, the program is somewhere in the middle of the received character. Since the data should be sampled every fourth timer tick, the tick counter is decremented and tested for zero. If non-zero no sample is needed and execution continues with XMIT. If zero, the tick counter is reset to four. Now the byte finished flag is tested to determine if the data sample is a data or stop bit. If reset, the sample is a data bit. The sample is done and the new bit rotated into the Rx deserializer. If this rotate

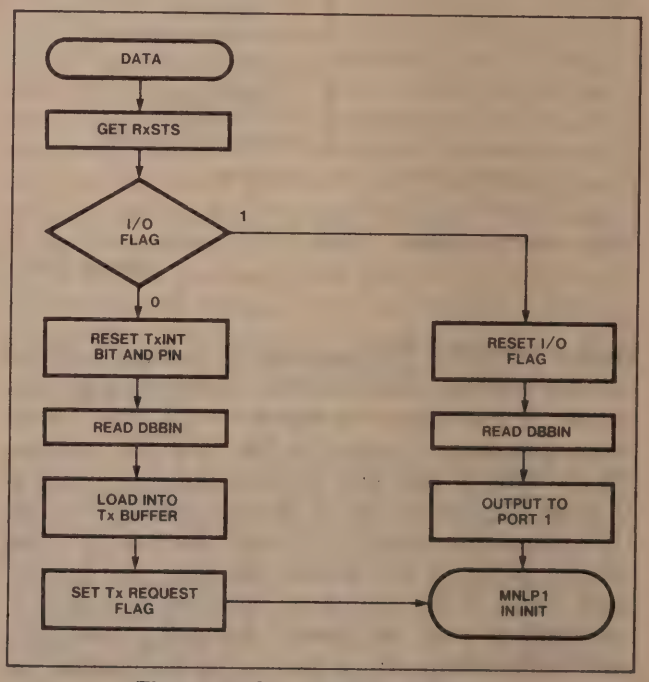


Figure 29C. Data Flow Chart

AFN-01536A

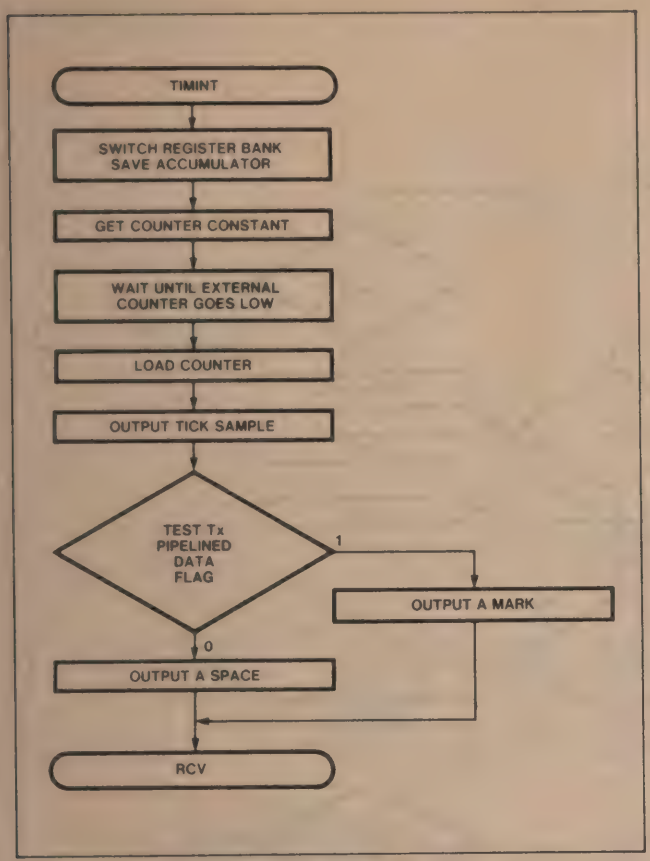


Figure 29D. TIMINT Flow Chart

sets the carry, that data bit was the last so the byte finished flag is set. If the carry is reset, the data bit is not the last so execution simply continues with XMIT.

Had the byte finished flag been set, this sample is for the stop bit. The RxD input is tested and if a space, the framing error flag is set. Otherwise, it is reset. Next, the Rx data ready flag is tested. If it is set, the master has not read the previous character so the overrun error flag is set. Then the Rx data ready flag is set and the received data character is transferred into the Rx holding register. The Rx, start bit, and byte finished flags are reset to get ready for the next character.

Execution of the transmitter routine, XMIT, follows the receiver, Figure 29F. The transmitter starts by checking the start bit flag in TxSTS. Recall that the actual transmit data is output at the beginning of the timer routine. The start bit flag indicates whether the current timer tick interrupt started the start bit. If it is set, the pipelined data output earlier in the routine was the start of the start bit so the flag is reset and the Tx tick counter is initialized. Nothing else is done this timer tick so the routine returns to the foreground.

If the start bit flag is reset, the Tx tick counter is incremented and tested. The test is performed modulo 4. If the counter mod 4 is not zero, it has not been four ticks since the transmitter was handled last so the routine simply returns. If the counter mod 4 is zero, it is time to handle the transmitter and the Tx flag is tested.

The Tx flag indicates whether the transmitter is active. If the transmitter is inactive, no character is currently being transmitted so the Tx request flag is tested to see if a new character is waiting in the Tx buffer. If no character is waiting (Tx request flag=0), the Tx interrupt pin and bit are set before returning to the foreground. If there is a character waiting, it is retrieved from the buffer and placed in the Tx serializer. The Tx request flag is reset while the Tx and start bit flags are set. A space is placed in the Tx pipelined data bit so a start bit will be output on the next tick. Since the Tx buffer is now empty, the Tx interrupt bit and pin are set to indicate the availability of the buffer to the master. The routine then returns to the foreground.

If the tick counter mod 4 is zero and the Tx flag indicates the transmitter is in the middle of a character, the tick counter is checked to see what transmitter operation is needed. If the counter is 28H (40D), all data bits plus the stop bits are complete. The character is therefore done and the Tx flag is reset. If the counter is 24H (36D), the data bits are complete and the next output should be a mark for the stop bit so a mark is loaded into the Tx pipelined data bit.

If neither of the above conditions are met for the counter, the transmitter is some place in the data field, so the next data bit is rotated out of the Tx serializer into the pipelined data bit. The next tick outputs this bit.

At this point the program execution is returned to the foreground.

That completes the discussion of the combination I/O device flow charts. The UPI software listing is shown in Appendix C1. Appendix C2 is example 8085A driver software.

Several observations concerning the drivers are appropriate. Notice that since the receiver and input port of the UPI use the OBF flag and interrupt output, the interrupt and flag are cleared when the master reads DBBOUT. This is not true for the transmitter. There is always some time after a master write of new transmitter data before the transmitter bit and pin are cleared. Thus in an interrupt-driven system, edge-sensitive interrupts should be

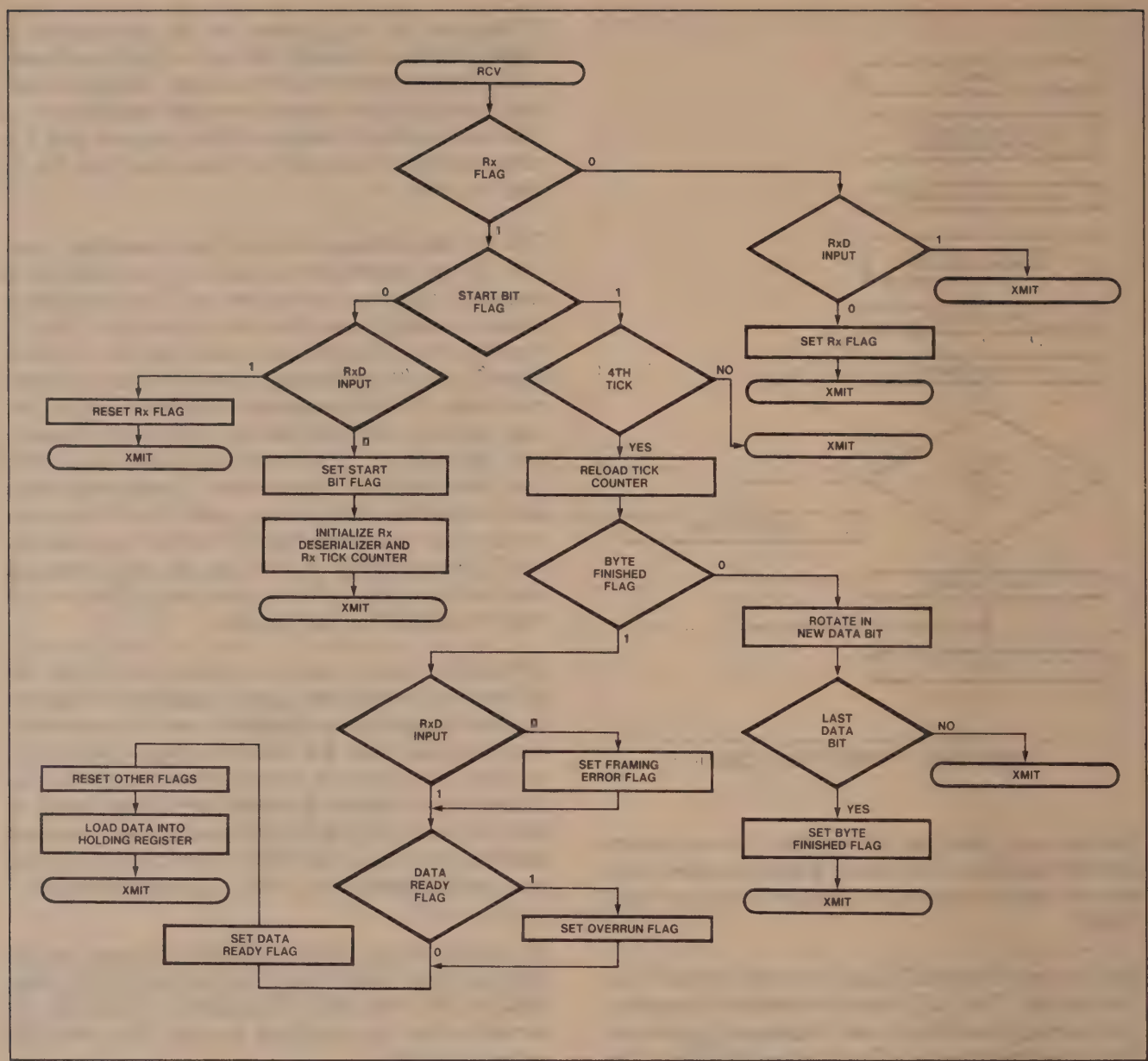


Figure 29E. RCV Flow Chart

used. For polled-systems, the software must wait after writing new data for IBF=0 before re-examining the Tx interrupt flag in STATUS.

Notice that this application uses none of the user data memory above Register Bank 1 and only 361 bytes of program memory. This leaves the door open for many improvements. Improvements that come to mind are increased buffering of the transmit or received data, modem control pins, and parallel port handshaking inputs.

This completes our discussion of specific UPI applications. Before concluding, let's look briefly at two debug techniques used during the development of these applications that you might find useful in your own designs.

DEBUG TECHNIQUES

Since the UPI is essentially a single-chip microcomputer, the classical data, address, and control buses are not available to the outside world during normal operation. This fact normally makes debugging a UPI design difficult; however, certain "tricks" can be included in the UPI software to ease this task.

If a UPI is handling multiple tasks, it is usually easier to code and debug each task individually. This is fairly standard procedure. Since each task usually utilizes only a subset of the total number of I/O pins,

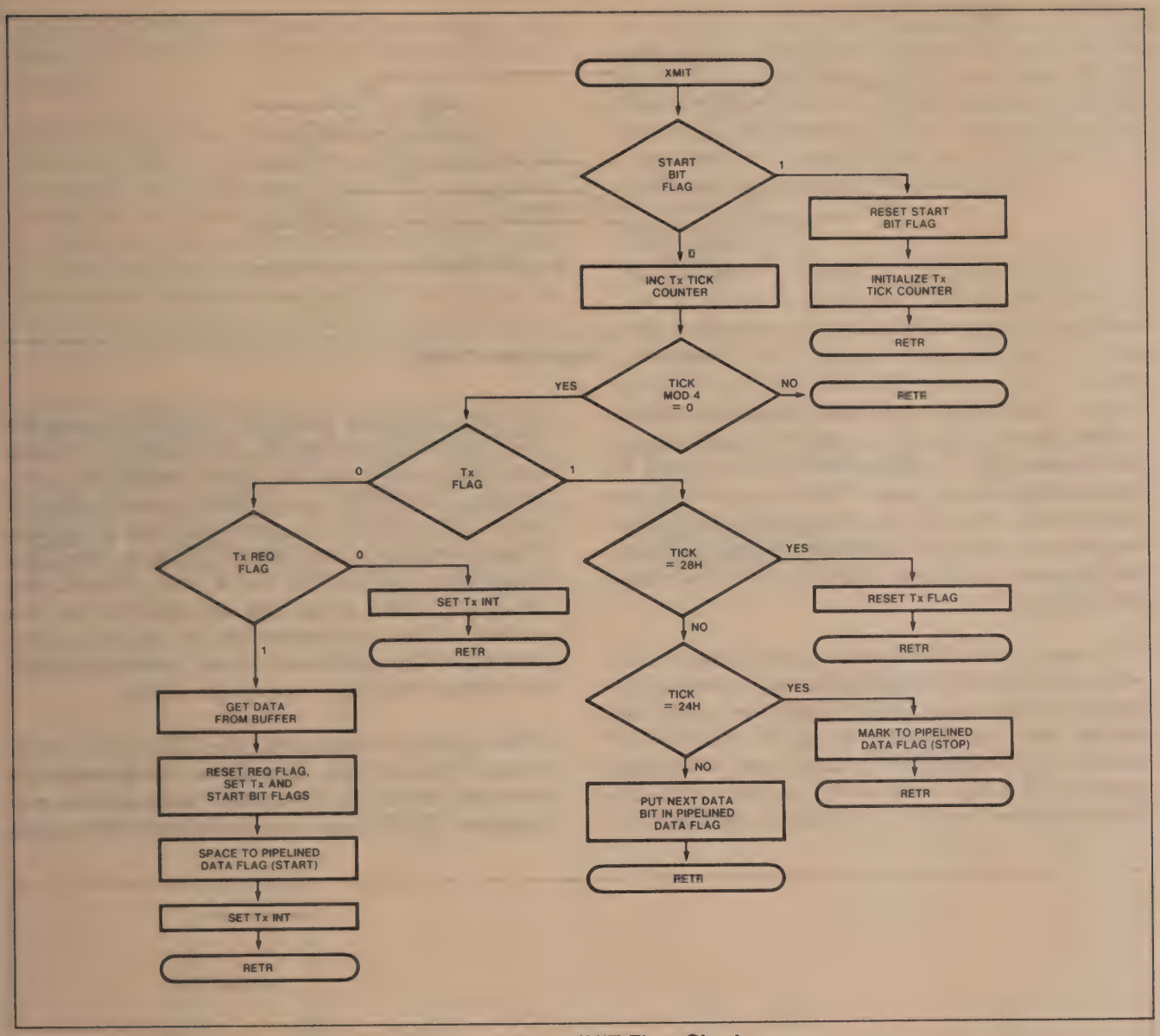


Figure 29F. XMIT Flow Chart

coding only one task leaves some I/O pins free. Port output instructions can then be added in the task code being debugged which toggle these unused pins to determine which section of task code is being executed at any particular time. The task can also be made to "wait" at various points by using an extra pin as an input and adding code to loop until a particular input condition is met.

One example of using an extra pin as an output is included in the combination serial/parallel device code. During initial development the receiver was not receiving characters correctly. Since this could be caused by incorrect sampling, three lines of code were added to toggle BIT 6 of PORT 2 at each tick of the sample clock. This code is at lines 184 and 185 of the listing. Thus by looking at the location of the tick sample pulse with respect to the received bit, the UPI sampling interval can be observed. The tick sample time was incorrect and the code was modified accordingly. Similar techniques could be applied at other locations in the program.

The EPROM version of the UPI (8741A) also contains another feature to aid in debug: the capability to single step thru a program. The user may step thru the program instruction-by-instruction. The address of the next instruction to be fetched is available on PORT 1 and the lower 2 bits of PORT 2. Figure 30 shows the timing used in the discussion below. When the single step input, $\overline{SS}$, is brought low, the internal processor responds by stopping during the fetch portion of the next instruction. This action is acknowledged by the processor raising the SYNC

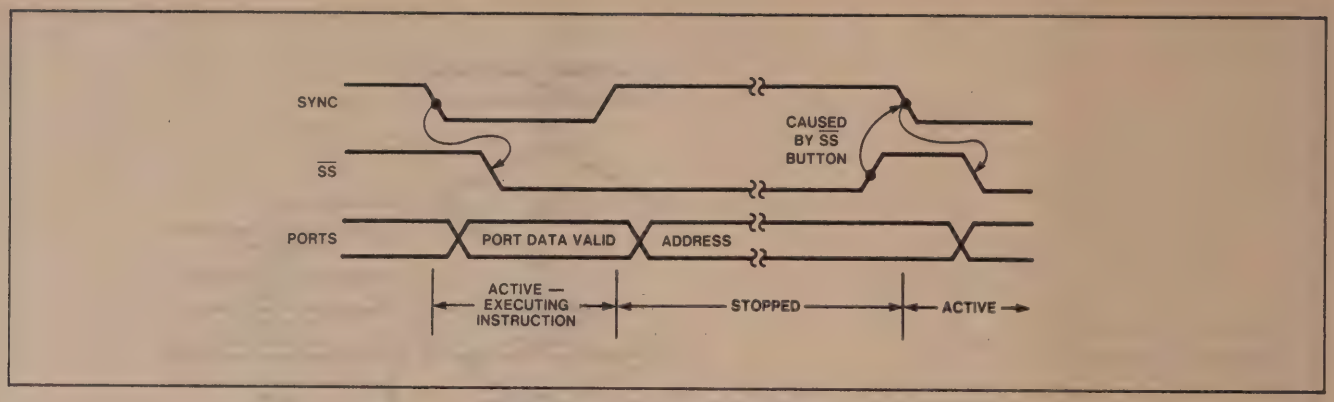


Figure 30. Single Step Timing

output. The address of the instruction to be fetched is then placed on the port pins. This state may be held indefinitely. To step to the next instruction, $\overline{SS}$ is raised high, which causes SYNC to go low, which is then used to return $\overline{SS}$ low. This allows the processor to advance to the next instruction. If $\overline{SS}$ is left high, the processor continues to execute at normal speed until $\overline{SS}$ goes low.

To preserve port functionality, port data is valid while SYNC is low. Figure 31 shows the external circuitry required to implement single step while preserving port functionality. S_1 is the RUN/STOP switch. When in the RUN position, the 7474 is held preset so $\overline{SS}$ is high and the UPI executes normally. When switched to STOP, the preset is removed and the next low-going transition of SYNC causes the 7474 to clear, lowering $\overline{SS}$. While sync is low, the port data is valid and the current instruction is executing. Low SYNC is also used to enable the tristate buffers when the ports are used as inputs. When execution is complete, SYNC goes high. This transition latches the valid port data in the 74LS374s. SYNC going high also signifies that the address of the next instruction will appear on the port pins. This state can be held indefinitely with the address data displayed on the LEDs.

When the S_2 is depressed, the 7474 is set which causes $\overline{SS}$ to go high. This allows the processor to fetch and execute the instruction whose address was displayed. SYNC going low during execution, clears

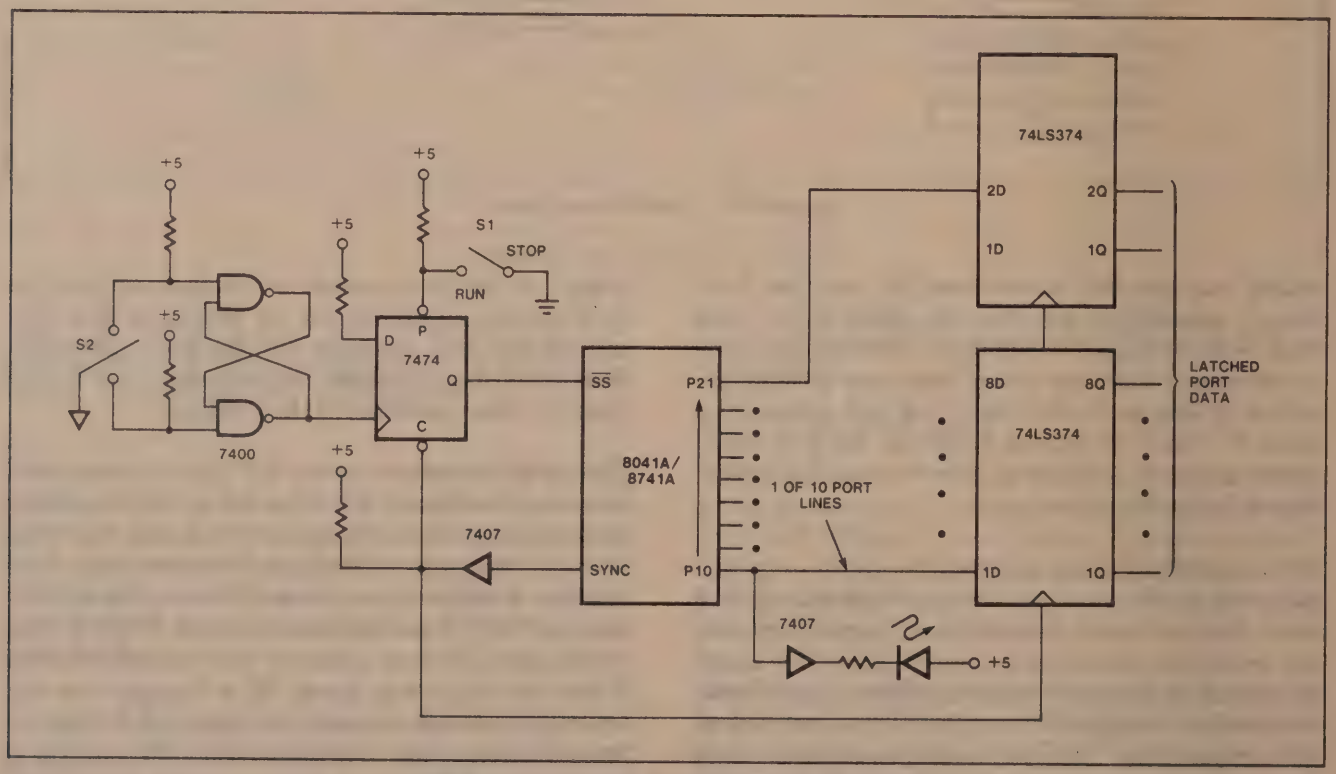


Figure 31. Single Step External Circuitry

the 7474 lowering $\overline{SS}$. Thus the processor again stops when execution is complete and the next fetch is started.

All UPI functions continue to operate while single stepping (the processor is actually executing NOPs internally while stopped). Both IBF and timer/counter interrupts can be serviced. The only change is that the interval timer is prescaled on single stepped instructions and, of course, will not indicate the correct intervals in real time. The total number of instruction which would have been executed during a given interval is the same however.

The single step circuitry can be used to step through a complete program; however, this might be a time-consuming job if the program is long or if only a portion is to be examined. The circuitry could easily be modified to incorporate the output toggling technique to determine when to run and stop. If you would like to step thru a particular section of code, an extra port pin could replace switch S_1. Extra instructions would then be added to lower the port when entering the code section and raise the port when exiting the section. The program would then stop when that section of code is reached allowing it to be stepped through. At the end of the section, the program would execute at normal speed.

CONCLUSION

Well, that's it. Machine readable (floppy disk or paper tape) source listings of UPI software for these applications are available in Insite, the Intel library of user-donated programs. Also available in Insite are the source listings for some of Intel's pre-programmed UPI products.

For information about Insite, write to:
Insite
Intel Corp.
3065 Bowers Ave.
Santa Clara, Ca 95051

APPENDIX AI

```
F1 ASM48 :F3:LED PRINT(:LP:) NOOBJECT


ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V3.0          PAGE    1

 LOC  OBJ        LINE        SOURCE STATEMENT

                   1 $MOD41A
                   2 ;        ***********************************************
                   3 ;        *    UPI-41A 8-DIGIT LED DISPLAY CONTROLLER    *
                   4 ;        ***********************************************
                   5 ;
                   6 ;
                   7 ;
                   8 ;THIS PROGRAM USES THE UPI-41A AS A LED DISPLAY CONTROLLER
                   9 ;WHICH SCANS AND REFRESHES EIGHT SEVEN-SEGMENT LED DISPLAYS.
                  10 ;THE CHARACTERS ARE DEFINED BY INPUT FROM A MASTER CPU IN THE
                  11 ;FORM OF ONE EIGHT BIT WORD PER DIGIT-CHARACTER SELECTION.
                  12 ;
                  13 ;
                  14 ;
                  15 ;*******************************************************************
                  16 ;
                  17 ;REGISTER DEFINITIONS:
                  18 ;        REGISTER                RB1                     RB0
                  19 ;        --------                ---                     ---
                  20 ;        R0              DISPLAY MAP POINTER             NOT USED
                  21 ;        R1              NOT USED                        NOT USED
                  22 ;        R2              DATA WORD AND CHARACTER STORAGE NOT USED
                  23 ;        R3              DIGIT COUNTER                   NOT USED
                  24 ;        R4              NOT USED                        NOT USED
                  25 ;        R5              NOT USED                        NOT USED
                  26 ;        R6              NOT USED                        NOT USED
                  27 ;        R7              ACCUMULATOR STORAGE             NOT USED
                  28 ;*******************************************************************
                  29 ;
                  30 ;PORT PIN DEFINITIONS:
                  31 ;        PIN             PORT 1 FUNCTION         PORT 2 FUNCTION
                  32 ;        ---             ---------------         ---------------
                  33 ;        P0-7            SEGMENT DRIVER CONTROL  DIGIT DRIVER CONTROL
                  34 ;
                  35 $EJECT
```

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V3.0              PAGE    2

 LOC  OBJ          LINE        SOURCE STATEMENT

                     36 ;*********************************************************************
                     37 ;DISPLAY DATA WORD BIT DEFINITION:
                     38 ;        BIT                 FUNCTION
                     39 ;        ---                 --------
                     40 ;        0-4                 CHARACTER SELECT
                     41 ;        5-7                 DIGIT SELECT
                     42 ;
                     43 ;CHARACTER SELECT:
                     44 ;                  D4  D3  D2  D1  D0  CHARACTER
                     45 ;                  0   0   0   0   0      0
                     46 ;                  0   0   0   0   1      1
                     47 ;                  0   0   0   1   0      2
                     48 ;                  0   0   0   1   1      3
                     49 ;                  0   0   1   0   0      4
                     50 ;                  0   0   1   0   1      5
                     51 ;                  0   0   1   1   0      6
                     52 ;                  0   0   1   1   1      7
                     53 ;                  0   1   0   0   0      8
                     54 ;                  0   1   0   0   1      9
                     55 ;                  0   1   0   1   0      A
                     56 ;                  0   1   0   1   1      B
                     57 ;                  0   1   1   0   0      C
                     58 ;                  0   1   1   0   1      D
                     59 ;                  0   1   1   1   0      E
                     60 ;                  0   1   1   1   1      F
                     61 ;                  1   0   0   0   0      .
                     62 ;                  1   0   0   0   1      G
                     63 ;                  1   0   0   1   0      H
                     64 ;                  1   0   0   1   1      I
                     65 ;                  1   0   1   0   0      J
                     66 ;                  1   0   1   0   1      L
                     67 ;                  1   0   1   1   0      N
                     68 ;                  1   0   1   1   1      O
                     69 ;                  1   1   0   0   0      P
                     70 ;                  1   1   0   0   1      R
                     71 ;                  1   1   0   1   0      T
                     72 ;                  1   1   0   1   1      U
                     73 ;                  1   1   1   0   0      Y
                     74 ;                  1   1   1   0   1      -
                     75 ;                  1   1   1   1   0      /
                     76 ;                  1   1   1   1   1      "BLANK"
                     77 ;
                     78 ;DIGIT SELECT:
                     79 ;                  D7  D6  D5   DIGIT NUMBER
                     80 ;                  0   0   0          1
                     81 ;                  0   0   1          2
                     82 ;                  0   1   0          3
                     83 ;                  0   1   1          4
                     84 ;                  1   0   0          5
                     85 ;                  1   0   1          6
                     86 ;                  1   1   0          7
                     87 ;                  1   1   1          8
                     88 ;*********************************************************************
                     89 $EJECT
```

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V3.0          PAGE    3

LOC  OBJ        LINE        SOURCE STATEMENT

                 90 ;*********************************************************************
                 91 ;                          EQUATES
                 92 ;THE FOLLOWING CODE DESIGNATES "TIME" AS A VARIABLE. THIS
                 93 ;ADJUSTS THE AMOUNT OF CYCLES THE TIMER COUNTS BEFORE
                 94 ;A TIMER INTERRUPT OCCURS AND REFRESHES THE DISPLAY. APPROXIMATELY
                 95 ;50 TIMES PER SECOND.
FFF1             96 TIME    EQU     -0FH     ;TIMER VALUE 2.5MSEC
                 97 ;*********************************************************************
                 98 ;                       INTERRUPT BRANCHING
                 99 ;THIS PORTION OF MEMORY IS DEDICATED FOR USE OF RESET AND
                100 ;INTERRUPT BRANCHING. WHEN THE INTERRUPTS ARE ENABLED THE
                101 ;CODE AT THE FOLLOWING DESIGNATED SPOTS ARE EXECUTED WHEN A
                102 ;RESET OR A INTERRUPT OCCURS.
0000            103         ORG     0            ;
0000 0409       104         JMP     START        ;RESET
0002 00         105         NOP                  ;
0003 0436       106         JMP     INPUT        ;IBF INTERRUPT
0005 00         107         NOP                  ;
0006 00         108         NOP                  ;
0007 041D       109         JMP     DISPLA       ;TIMER INTERRUPT
                110 ;*********************************************************************
                111 ;                       INITIALIZATION
                112 ;THE FOLLOWING CODE SETS UP THE UPI-41 AND DISPLAY HARDWARE
                113 ;INTO OPERATIONAL FORMAT. THE DISPLAY IS TURNED OFF, THE DISPLAY
                114 ;MAP IS FILLED WITH "BLANK" CHARACTERS, THE TIMER SET AND THE
                115 ;INTERRUPTS ARE ENABLED.
                116 ;
0009 D5         117 START:  SEL     RB1          ;
000A 8A08       118         ORL     P2,#08H      ;TURN DIGIT DRIVERS OFF
000C B838       119         MOV     R0,#38H      ;DISPLAY MAP POINTER,BOTTOM OF DISPLAY MAP
000E 23FF       120 BLKMAP: MOV     A,#0FFH      ;FF="BLANK"
0010 A0         121         MOV     @R0,A        ;BLANK TO DISPLAY MAP
0011 18         122         INC     R0           ;INCREMENT DISPLAY MAP POINTER
0012 F8         123         MOV     A,R0         ;DISPLAY MAP POINTER TO ACCUMULATOR
0013 B20E       124         JB5     BLKMAP       ;BLANK DISPLAY MAP TILL FILLED
0015 BB00       125         MOV     R3,#00H      ;SET DIGIT COUNTER TO 0
0017 23F1       126         MOV     A,#TIME      ;TIMER VALUE
0019 62         127         MOV     T,A          ;LOAD TIMER
001A 55         128         STRT    T            ;START TIMER
001B 25         129         EN      TCNTI        ;ENABLE TIMER INTERRUPT
001C 05         130         EN      I            ;ENABLE IBF INTERRUPT
                131 ;*********************************************************************
                132 ;                       USER PROGRAM
                133 ;A USERS PROGRAM WOULD INITIALIZE AT THIS POINT. THE FOLLOWING
                134 ;CODE IS UND CONCLUDED WITH
                135 ;SYNC CHARACTERS (0AAH).  A CHECKSUM BYTE IMMEDIATELY PRECEEDS THE
                136 ;FINAL SYNC.  WHEN READING, THE CONTROLLE*************************************
                137 $EJECT
```

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V3.0          PAGE    4

 LOC  OBJ          LINE        SOURCE STATEMENT

                    138 ;*********************************************************************
                    139 ;                          DISPLAY ROUTINE
                    140 ; THIS PORTION OF THIS PROGRAM IS AN INTERRUPT ROUTINE WHICH IS
                    141 ;ACTED UPON WHEN THE TIMER COUNT IS COMPLETED.  THE ROUTINE UPDATES
                    142 ;ONE DISPLAY DIGIT FROM THE DISPLAY MAP PER INTERRUPT SEQUENTIALLY,
                    143 ;THUS EIGHT TIMER INTERRUPTS WILL HAVE REFRESHED THE ENTIRE DISPLAY.
                    144 ;REGISTER BANK 1 IS SELECTED AND THE ACCUMULATOR IS SAVED UPON
                    145 ;ENTERING THE ROUTINE.  ONCE THE DISPLAY HAS BEEN REFRESHED THE TIMER
                    146 ;IS RESET AND THE ACCUMULATOR AND PRE-INTERRUPT REGISTER BANK IS RESTORED.
                    147 ;
 001D D5            148 DISPLA: SEL     RB1          ;REGISTER BANK 1
 001E AF            149         MOV     R7,A         ;SAVE ACCUMULATOR
 001F 8A08          150         ORL     P2,#08H      ;TURN DIGIT DRIVERS OFF
 0021 FB            151         MOV     A,R3         ;DIGIT COUNTER TO ACCUMULATOR
 0022 4338          152         ORL     A,#38H       ;"OR" TO GET DISPLAY MAP ADDRESS
 0024 A8            153         MOV     R0,A         ;DISPLAY MAP POINTER
 0025 F0            154         MOV     A,@R0        ;GET CHARACTER FROM DISPLAY MAP
 0026 39            155         OUTL    P1,A         ;OUTPUT CHARACTER TO SEGMENT DRIVERS
 0027 FB            156         MOV     A,R3         ;DIGIT COUNTER VALUE TO ACCUMULATOR
 0028 3A            157         OUTL    P2,A         ;OUTPUT TO DIGIT DRIVERS
 0029 1B            158         INC     R3           ;INCREMENT DIGIT COUNTER
 002A D307          159         XRL     A,#07H       ;CHECK IF AT LAST DIGIT
 002C 9630          160         JNZ     SETIME       ;RESET TIMER IN NOT LAST DIGIT
 002E BB00          161         MOV     R3,#00H      ;RESET DIGIT COUNTER
 0030 23F1          162 SETIME: MOV     A,#TIME      ;TIMER VALUE
 0032 62            163         MOV     T,A          ;LOAD TIMER
 0033 55            164         STRT    T            ;START TIMER
 0034 FF            165         MOV     A,R7         ;RESTORE ACCUMULATOR
 0035 93            166         RETR                 ;RETURN
                    167 ;*********************************************************************
                    168 $EJECT
```

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V3.0              PAGE    5

 LOC  OBJ          LINE        SOURCE STATEMENT

                    169 ;
                    170 ;*********************************************************************
                    171 ;                    INPUT CHARACTER AND DIGIT ROUTINE
                    172 ;  THIS PORTION OF THE PROGRAM IS AN INTERRUPT ROUTINE WHICH
                    173 ;IS ACTED UPON WHEN THE IBF BIT IS SET. THE ROUTINE GETS THE
                    174 ;DISPLAY DATA WORD FROM THE DBB AND DEFINES BOTH THE DIGIT AND
                    175 ;THE CHARACTER TO BE DISPLAYED. THIS IS DONE BY MEANS OF A
                    176 ;CHARACTER LOOP-UP TABLE AND A DISPLAY MAP FOR DIGIT AND CHARACTER
                    177 ;LOCATION. SPECIAL CONSIDERATION IS TAKEN FOR A DECIMAL POINT WHICH IS
                    178 ;SIMPLY ADDED TO THE EXISTING CHARACTER IN THE DISPLAY MAP. REGISTER
                    179 ;BANK 1 IS SELECTED AND THE ACCUMULATOR IS SAVED UPON ENTERING
                    180 ;THE ROUTINE. ONCE THE DATA WORD HAS BEEN FULLY DEFINED THE ACCUMULATOR
                    181 ;AND THE PRE-INTERRUPT REGISTER BANK IS RESTORED.
                    182 ;
 0036 D5            183 INPUT:  SEL     RB1          ;REGISTER BANK 1
 0037 AF            184         MOV     R7,A         ;SAVE ACCUMULATOR
 0038 22            185         IN      A,DBB        ;GET DATA
 0039 AA            186         MOV     R2,A         ;SAVE DATA WORD
 003A 47            187         SWAP    A            ;DEFINE DIGIT LOCATION
 003B 77            188         RR      A            ;
 003C 5307          189         ANL     A,#07H       ;
 003E 4338          190         ORL     A,#38H       ;
 0040 A8            191         MOV     R0,A         ;DIGIT LOCATION IN DIGIT POINTER
 0041 FA            192         MOV     A,R2         ;SAVED DATA WORD TO ACCUMULATOR
 0042 531F          193         ANL     A,#1FH       ;DEFINE CHARACTER LOOK-UP-TABLE LOC.
 0044 E3            194         MOVP3   A,@A         ;GET CHARACTER
 0045 AA            195         MOV     R2,A         ;SAVE CHARACTER
 0046 D37F          196         XRL     A,#7FH       ;IS CHARACTER DECIMAL POINT
 0048 C64E          197         JZ      DPOINT       ;
 004A FA            198         MOV     A,R2         ;SAVED CHARACTER TO ACCUMULATOR
 004B A0            199         MOV     @R0,A        ;CHARACTER TO DISPLAY MAP
 004C 0451          200         JMP     RETURN       ;
 004E FA            201 DPOINT: MOV     A,R2         ;SAVED CHARACTER TO ACCUMULATOR
 004F 50            202         ANL     A,@R0        ;"AND" WITH OLD CHARACTER
 0050 A0            203         MOV     @R0,A        ;BACK TO DISPLAY MAP
 0051 FF            204 RETURN: MOV     A,R7         ;RESTORE ACCUMULATOR
 0052 93            205         RETR
                    206 ;*********************************************************************
                    207 $EJECT
```

APPENDIX AII

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V3.0              PAGE    6

 LOC  OBJ          LINE        SOURCE STATEMENT

                    208 ;*********************************************************************
                    209 ;                          LOOK-UP TABLE
                    210 ; THIS LOOK-UP TABLE ORIGINATES IN PAGE 3 OF THE UPI-41 PROGRAM
                    211 ;MEMORY. IT IS USED TO DEFINE THE CORRECT LEVEL OF EACH SEGMENT
                    212 ;AND DECIMAL POINT FOR A SELECTED CHARACTER FROM THE INPUT ROUTINE.
                    213 ;INVERSE LOGIC IS USED BECAUSE OF THE SPECIFIC DRIVER CIRCUITRY, THUS
                    214 ;A 1 ON A GIVEN SEGMENT MEANS IT IS OFF AND A 0 MEANS IT IS ON.
                    215 ;
                    216                                   ;*******SEGMENTS********
 0300               217          ORG     300H             ;DP  G  F  E  D  C  B  A
 0300 C0            218 CH0:     DB      0C0H             ;1   1  0  0  0  0  0  0
 0301 F9            219 CH1:     DB      0F9H             ;1   1  1  1  1  0  0  1
 0302 A4            220 CH2:     DB      0A4H             ;1   0  1  0  0  1  0  0
 0303 B0            221 CH3:     DB      0B0H             ;1   0  1  1  0  0  0  0
 0304 99            222 CH4:     DB      99H              ;1   0  0  1  1  0  0  1
 0305 92            223 CH5:     DB      92H              ;1   0  0  1  0  0  1  0
 0306 82            224 CH6:     DB      82H              ;1   0  0  0  0  0  1  0
 0307 F8            225 CH7:     DB      0F8H             ;1   1  1  1  1  0  0  0
 0308 80            226 CH8:     DB      80H              ;1   0  0  0  0  0  0  0
 0309 98            227 CH9:     DB      98H              ;1   0  0  1  1  0  0  0
 030A 88            228 CHA:     DB      88H              ;1   0  0  0  1  0  0  0
 030B 83            229 CHB:     DB      83H              ;1   0  0  0  0  0  1  1
 030C C6            230 CHC:     DB      0C6H             ;1   1  0  0  0  1  1  0
 030D A1            231 CHD:     DB      0A1H             ;1   0  1  0  0  0  0  1
 030E 86            232 CHE:     DB      86H              ;1   0  0  0  0  1  1  0
 030F 8E            233 CHF:     DB      8EH              ;1   0  0  0  1  1  1  0
 0310 7F            234 CHDP:    DB      7FH              ;0   1  1  1  1  1  1  1
 0311 C2            235 CHG:     DB      0C2H             ;1   1  0  0  0  0  1  0
 0312 89            236 CHH:     DB      89H              ;1   0  0  0  1  0  0  1
 0313 FB            237 CHI:     DB      0FBH             ;1   1  1  1  1  0  1  1
 0314 E1            238 CHJ:     DB      0E1H             ;1   1  1  0  0  0  0  1
 0315 C7            239 CHL:     DB      0C7H             ;1   1  0  0  0  1  1  1
 0316 AB            240 CHN:     DB      0ABH             ;1   0  1  0  1  0  1  1
 0317 A3            241 CHO:     DB      0A3H             ;1   0  1  0  0  0  1  1
 0318 8C            242 CHP:     DB      8CH              ;1   0  0  0  1  1  0  0
 0319 AF            243 CHR:     DB      0AFH             ;1   0  1  0  1  1  1  1
 031A 87            244 CHT:     DB      87H              ;1   0  0  0  0  1  1  1
 031B C1            245 CHU:     DB      0C1H             ;1   1  0  0  0  0  0  1
 031C 91            246 CHY:     DB      91H              ;1   0  0  1  0  0  0  1
 031D BF            247 CHDASH:  DB      0BFH             ;1   0  1  1  1  1  1  1
 031E FD            248 CHAPOS:  DB      0FDH             ;1   1  1  1  1  1  0  1
 031F FF            249 BLANK:   DB      0FFH             ;1   1  1  1  1  1  1  1
                    250 ;*********************************************************************
                    251          END

USER SYMBOLS
BLANK  031F   BLKMAP 000E   CH0    0300   CH1    0301   CH2    0302   CH3    0303   CH4    0304   CH5    0305
CH6    0306   CH7    0307   CH8    0308   CH9    0309   CHA    030A   CHAPOS 031E   CHB    030B   CHC    030C
CHD    030D   CHDASH 031D   CHDP   0310   CHE    030E   CHF    030F   CHG    0311   CHH    0312   CHI    0313
CHJ    0314   CHL    0315   CHN    0316   CHO    0317   CHP    0318   CHR    0319   CHT    031A   CHU    031B
CHY    031C   DISPLA 001D   DPOINT 004E   INPUT  0036   RETURN 0051   SETIME 0030   START  0009   TIME   FFF1

ASSEMBLY COMPLETE,    NO ERRORS
```

```
LOC  OBJ        SEQ        SOURCE STATEMENT

                 1 ;
                 2 ;8085A SUBROUTINE TO DISPLAY THE 8-DIGIT BUFFER STARTING
                 3 ;AT THE LOCATION POINTED AT BY MSGSRT ON THE UPI-CONTROLLED
                 4 ;LED DISPLAY.
                 5 ;
                 6 ;INPUTS:MSGSRT - MESSAGE START LOCATION POINTER
                 7 ;DESTROYS: A, F/F'S
                 8 ;CALLS: OUTCHR
                 9 ;
4000             10        ORG     4000H
00E5             11 STATUS EQU     0E5H          ;UPI STATUS PORT
0002             12 IBF    EQU     02H           ;UPI IBF FLAG MASK
00E4             13 DBBIN  EQU     0E4H          ;UPI DBBIN PORT
                 14 ;
4000 E5          15 DSPLAY: PUSH   H             ;SAVE HL
4001 C5          16         PUSH   B             ;SAVE BC
4002 2A2840      17         LHLD   MSGSRT        ;LOAD HL WITH MESSAGE START ADR
4005 0600        18         MVI    B,00H         ;INITIALIZE DIGIT COUNTER
4007 7E          19 S1:     MOV    A,M           ;GET CHR FROM BUFFER
4008 E61F        20         ANI    1FH           ;MAKE IT 5 BITS
400A 80          21         ADD    B             ;ADD IN DIGIT COUNTER
400B 4F          22         MOV    C,A           ;SAVE TOTAL IN C
400C CD1D40      23         CALL   OUTCHR        ;OUTPUT CHR PLUS LOCATION TO UPI
400F 78          24         MOV    A,B           ;GET DIGIT COUNTER
4010 C620        25         ADI    20H           ;INC FOR NEXT DIGIT
4012 DA1A40      26         JC     EXIT          ;DONE IF CARRY SET
4015 47          27         MOV    B,A           ;RESTORE DIGIT COUNTER
4016 23          28         INX    H             ;INC MESSAGE POINTER
4017 C30740      29         JMP    S1            ;GO GET NEXT CHR
                 30 ;
401A C1          31 EXIT:   POP    B             ;RESTORE BC
401B E1          32         POP    H             ;RESTORE HL
401C C9          33         RET                  ;RETURN
                 34 ;
                 35 ;SUBROUTINE TO OUTPUT CHR TO UPI
                 36 ;
401D DBE5        37 OUTCHR: IN     STATUS        ;READ UPI STATUS
401F E602        38         ANI    IBF           ;LOOK AT IBF
4021 C21D40      39         JNZ    OUTCHR        ;WAIT UNTIL IBF=0
4024 79          40         MOV    A,C           ;GET CHR
4025 D3E4        41         OUT    DBBIN         ;OUTPUT CHR TO UPI DBBIN
4027 C9          42         RET                  ;RETURN
                 43 ;
0002             44 MSGSRT: DS     02H           ;LOCATION OF MESSAGE START POINTER
                 45 ;
                 46 END
```

```
F1 ASM48 .F3 SENSOR NOOBJECT PRINT(:LP:)


ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V3.0              PAGE    1

 LOC  OBJ          LINE        SOURCE STATEMENT

                     1 $MOD41A
                     2 ;          ********************************************
                     3 ;          *      UPI-41A SENSOR MATRIX CONTROLLER     *
                     4 ;          ********************************************
                     5 ;
                     6 ;          THIS PROGRAM USES THE UPI-41A AS A SENSOR MATRIX CONTROLLER
                     7 ;IT HAS MONITORING CAPABILITIES OF UP TO 128 SENSORS.  THE COORDINATE
                     8 ;AND SENSOR STATUS OF EACH DETECTED CHANGE IS AVAILABLE TO THE MASTER
                     9 ;MICROPROCESSOR IN A SINGLE BYTE.  A 40X8 FIFO QUEUE IS PROVIDED FOR
                    10 ;DATA BUFFERING.  BOTH HARDWARE OR POLLED INTERRUPT METHODS CAN BE USED
                    11 ;TO NOTIFY THE MASTER OF A DETECTED SENSOR CHANGE.
                    12 ;
                    13 ;*********************************************************************
                    14 ;
                    15 ;REGISTER DEFINITIONS:
                    16 ;        REGISTER              RB0                    RB1
                    17 ;        --------              ---                    ---
                    18 ;          R0            MATRIX MAP POINTER         NOT USED
                    19 ;          R1            FIFO POINTER               NOT USED
                    20 ;          R2            SCAN ROW SELECT            NOT USED
                    21 ;          R3            COLUMN COUNTER             NOT USED
                    22 ;          R4            FIFO-IN                    NOT USED
                    23 ;          R5            FIFO-OUT                   NOT USED
                    24 ;          R6            CHANGE WORD                NOT USED
                    25 ;          R7            COMPARE                    NOT USED
                    26 ;
                    27 ;*********************************************************************
                    28 ;
                    29 ;PORT PIN DEFINITIONS:
                    30 ;
                    31 ;PIN       PORT 1 FUNCTION     PIN        PORT 2 FUNCTION
                    32 ;---       ---------------     ---        ---------------
                    33 ;P0-7      COLUMN LINE INPUTS  P0-3       ROW SELECT OUTPUTS
                    34 ;                              P4         FIFO NOT EMPTY INTERRUPT
                    35 ;                              P5         OBF INTERRUPT
                    36 ;                              P6-7       NOT USED
                    37 ;
                    38 ;*********************************************************************
                    39
                    40 $EJECT
```

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V3.0              PAGE    2

 LOC  OBJ          LINE        SOURCE STATEMENT

                     41 ;*********************************************************************
                     42 ;
                     43 ;CHANGE WORD BIT DEFINITION:
                     44 ;
                     45 ;                    BIT                    FUNCTION
                     46 ;                    ---                    --------
                     47 ;                    D0-6                   SENSOR COORDINATE
                     48 ;                    D7                     SENSOR STATUS
                     49 ;
                     50 ;*********************************************************************
                     51 ;
                     52 ;STATUS REGISTER BIT DEFINITION:
                     53 ;
                     54 ;                    BIT                    FUNCTION
                     55 ;                    ---                    --------
                     56 ;                    D0                     OBF
                     57 ;                    D1-3                   IBF, F0, F1 (NOT USED)
                     58 ;                    D4                     FIFO NOT EMPTY
                     59 ;                    D5-7                   USED DEFINED (NOT USED)
                     60 ;
                     61 ;*********************************************************************
                     62 ;
                     63 ;                          EQUATES
                     64 ;
                     65 ;THE FOLLOWING CODE DESIGNATES THREE VARIABLES; SCANTM,FIFOBA
                     66 ;AND FIFOTA.  SCANTM ADJUSTS THE LENGTH OF A DELAY BETWEEN
                     67 ;SCANNING SWITCH.  THIS SIMULATES DEBOUNCE FUNCTIONS.  FIFOBA
                     68 ;IS THE BOTTOM ADDRESS OF THE FIFO.  FIFOTA IS THE TOP ADDRESS
                     69 ;OF THE FIFO.  THIS MAKES IT POSSIBLE TO HAVE A FIFO 3 TO 40
                     70 ;BYTES IN LENGTH.
                     71 ;
                     72 ;*********************************************************************
                     73 ;
 000F                74 SCANTM  EQU    0FH            ;SCAN TIME ADJUST
 0008                75 FIFOBA  EQU    08H            ;FIFO BOTTOM ADDRESS
 002F                76 FIFOTA  EQU    2FH            ;FIFO TOP ADDRESS
                     77
                     78 $EJECT
```

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V3.0          PAGE    3

 LOC  OBJ          LINE        SOURCE STATEMENT

                     79 ;*********************************************************************
                     80 ;
                     81 ;                              INITIALIZATION
                     82 ;
                     83 ;THE PROGRAM STARTS AT THE FOLLOWING CODE UPON RESET.  WITHIN
                     84 ;THIS INITIALIZATION SECTION THE REGISTERS THAT MAINTAIN THE MATRIX
                     85 ;MAP,FIFO AND ROW SCANNING ARE SET UP.  PORT 1 IS SET HIGH FOR USE
                     86 ;AS AN INPUT PORT FOR THE COLUMN STATUS.  BIT 4 OF STATUS REGISTER IS
                     87 ;WRITTEN TO CONVEY A FIFO EMPTY CONDITION.  THE INITIAL COLUMN STATUS
                     88 ;OF ALL THE ROWS IN THE SENSOR MATRIX IS THEN READ INTO THE MATRIX
                     89 ;MAP.  ONCE THE MATRIX MAP IS FILLED THE OBF INTERRUPT (PORT 2-4) IS
                     90 ;ENABLED.
                     91 ;
                     92 ;*********************************************************************
                     93 ;
 0000                94          ORG     0
 0000 B83F           95 INITMX:  MOV     R0,#3FH        ;MATRIX MAP POINTER REGISTER,TOP ADDRESS
 0002 BA0F           96          MOV     R2,#0FH        ;SCAN ROW SELECT REGISTER,TOP ROW
 0004 BC08           97          MOV     R4,#FIFOBA     ;FIFO INPUT ADDRESS REGISTER,BOTTOM OF FIFO
 0006 BD2F           98          MOV     R5,#FIFOTA     ;FIFO OUTPUT ADDRESS REGISTER,TOP OF FIFO
 0008 89FF           99          ORL     P1,#0FFH       ;INITIALIZE PORT 1 HIGH FOR INPUTS
 000A 2300          100          MOV     A,#00H         ;INITIALIZE STATUS REGISTER, FIFO EMPTY
 000C 90            101          MOV     STS,A          ;WRITE TO STATUS REGISTER, BITS 4-7
 000D FA            102 FILLMX:  MOV     A,R2           ;SCAN ROW SELECT TO ACCUMULATOR
 000E 3A            103          OUTL    P2,A           ;OUTPUT SCAN ROW SELECT TO PORT 2
 000F 09            104          IN      A,P1           ;INPUT COLUMN STATUS PORT 1
 0010 A0            105          MOV     @R0,A          ;LOAD MATRIX MAP WITH COLUMN STATUS
 0011 FA            106          MOV     A,R2           ;CHECK SCAN ROW SELECT REGISTER VALUE FOR 0
 0012 C618          107          JZ      OBFINT         ;IF 0 ENABLE OBF INTERRUPT
 0014 C8            108          DEC     R0             ;DECREMENT TO NEXT MATRIX MAP ADDRESS
 0015 CA            109          DEC     R2             ;DECREMENT TO SCAN NEXT ROW
 0016 040D          110          JMP     FILLMX         ;FILL NEXT MATRIX MAP ADDRESS
 0018 BA10          111 OBFINT:  MOV     R2,#10H        ;BIT 4 HIGH IN ROW SCAN SELECT REGISTER
 001A FA            112          MOV     A,R2           ;ROW SCAN SELECT VALUE TO ACCUMULATOR
 001B 3A            113          OUTL    P2,A           ;INITIALIZE PORT 2, BIT 4 FOR "EN FLAGS"
 001C F5            114          EN      FLAGS          ;ENABLE OBF INTERRUPT PORT 2, BIT 4
                    115
                    116 $EJECT
```

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V3.0          PAGE    4

 LOC  OBJ          LINE        SOURCE STATEMENT

                    117 ;*********************************************************************
                    118 ;
                    119 ;                          SCAN AND COMPARE
                    120 ;
                    121 ;THE FOLLOWING CODE IS THE SCAN AND COMPARE SECTION OF THE PROGRAM.
                    122 ;UPON ENTERING THIS SECTION A CHECK IS MADE TO SEE IF THE ENTIRE MATRIX
                    123 ;HAS BEEN SCANNED.   IF SO THE REGISTERS THAT MAINTAIN THE MATRIX MAP AND ROW
                    124 ;SCANNING ARE RESET TO THE BEGINNING OF THE SENSOR MATRIX.   IF THE ENTIRE
                    125 ;MATRIX HASNT BEEN SCANNED THE REGISTERS INCREMENT TO SCAN THE NEXT ROW.
                    126 ;FROM THIS POINT ON THE ROW SCAN SELECT REGISTER IS USED FOR TWO FUNCTIONS.
                    127 ;BITS 0-3 FOR SCANNING AND BITS 4 AND 5 FOR THE EXTERNAL INTERRUPTS.   THUSLY
                    128 ;ALL USAGE OF THE REGISTERS IS DONE BY LOGICALLY MASKING IT SO AS TO ONLY
                    129 ;AFFECT THE FUNCTION DESIRED.   ONCE THE REGISTERS ARE RESET, ONE ROW OF THE
                    130 ;SENSOR MATRIX IS SCANNED.   A DELAY IS EXECUTED TO ADJUST FOR SCAN TIME
                    131 ;(DEBOUNCE).   A BYTE OF COLUMN STATUS IS THEN READ INTO THE MATRIX MAP.
                    132 ;AT THE TIME THE NEW COLUMN STATUS IS COMPARED TO THE OLD.   THE RESULT IS
                    133 ;STORED IN THE COMPARE REGISTER.   THE PROGRAM IS THEN ROUTED ACCORDING TO
                    134 ;WHETHER OR NOT A CHANGE WAS DETECTED.
                    135 ;
                    136 ;*********************************************************************
                    137
 001D FA            138 ADJREQ: MOV     A,R2          ;SCAN ROW SELECT TO ACCUMULATOR
 001E 530F          139         ANL     A,#0FH        ;CHECK FOR 0 SCAN VALUE ONLY,NOT INTERRUPT
 0020 C626          140         JZ      RSETRQ        ;IF 0 RESET REGISTERS
 0022 C8            141         DEC     R0            ;DECREMENT MATRIX MAP POINTER
 0023 CA            142         DEC     R2            ;DECREMENT SCAN ROW SELECT
 0024 042C          143         JMP     SCANMX        ;SCAN MATRIX
 0026 B83F          144 RSETRQ: MOV     R0,#3FH       ;RESET MATRIX MAP POINTER REGISTER,TOP ADDRESS
 0028 FA            145         MOV     A,R2          ;SCAN ROW SELECT TO ACCUMULATOR
 0029 430F          146         ORL     A,#0FH        ;RESET SCAN ROW SELECT,NO INTERRUPT CHANGE
 002B AA            147         MOV     R2,A          ;SCAN ROW SELECT REGISTER
 002C FA            148 SCANMX: MOV     A,R2          ;SCAN ROW SELECT TO ACCUMULATOR
 002D 3A            149         OUTL    P2,A          ;OUTPUT SCAN ROW SELECT TO PORT 2
 002E BB0F          150         MOV     R3,#SCANTM    ;SET DELAY FOR OUTPUT SCAN TIME
 0030 EB30          151 DELAY2: DJNZ    R3,DELAY2     ;DELAY
 0032 09            152         IN      A,P1          ;INPUT COLUMN STATUS FROM PORT 1 TO ACCUMULATOR
 0033 20            153         XCH     A,@R0         ;STORE NEW COLUMN STATUS SAVE OLD IN ACCUMULATOR
 0034 D0            154         XRL     A,@R0         ;COMPARE OLD WITH NEW COLUMN STATUS
 0035 AF            155         MOV     R7,A          ;SAVE COMPARE RESULT IN COMPARE REGISTER
 0036 C669          156         JZ      CHFFUL        ;IF THE SAME, CHECK IF FIFO IS FULL
                    157
                    158 $EJECT
```

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V3.0          PAGE    5

 LOC  OBJ        LINE        SOURCE STATEMENT

                  159 ;*********************************************************************
                  160 ;
                  161 ;                        CHANGE WORD ENCODING
                  162 ;
                  163 ;THE FOLLOWING CODE IS THE CHANGE WORD ENCODING SECTION.  THIS
                  164 ;SECTION IS ONLY EXECUTED IF A CHANGE WAS DETECTED.  THE COLUMN COUNTER
                  165 ;IS SET AND DECREMENTED TO DESIGNATE EACH OF THE 8 COLUMNS.  THE COMPARE
                  166 ;REGISTER IS LOOKED AT ONE BIT AT A TIME TO FIND THE EXACT LOCATION OF
                  167 ;THE CHANGE(S).  WHEN A CHANGE IS FOUND IT IS ENCODED BY GIVING IT A
                  168 ;COORDINATE FOR ITS LOCATION.  THIS IS DONE BY COMBINING THE PRESENT VALUE
                  169 ;IN THE ROW SCAN SELECT REGISTER AND THE COLUMN COUNTER.  THE ACTUAL STATUS
                  170 ;OF THAT SENSOR IS ESTABLISHED BY LOOKING AT THE CORRESPONDING BYTE IN
                  171 ;THE MATRIX MAP.  THIS STATUS IS COMBINED WITH THE COORDINATE TO ESTABLISH
                  172 ;THE CHANGE WORD.  THE CHANGE WORD IS THEN STORED IN THE CHANGE WORD REGISTER
                  173 ;
                  174 ;*********************************************************************
                  175 ;
 0038 B808        176          MOV     R3,#08H        ;SET COLUMN COUNTER REGISTER TO 8
 003A CB          177 RRLOOK:  DEC     R3             ;DECREMENT COLUMN COUNTER
 003B F0          178          MOV     A,@R0          ;COLUMN STATUS TO ACCUMULATOR
 003C 77          179          RR      A              ;ROTATE COLUMN STATUS RIGHT
 003D A0          180          MOV     @R0,A          ;ROTATED COLUMN STATUS BACK TO MATRIX MAP
 003E FF          181          MOV     A,R7           ;COMPARE REGISTER VALUE TO ACCUMULATOR
 003F 77          182          RR      A              ;ROTATE COMPARE VALUE RIGHT
 0040 AF          183          MOV     R7,A           ;ROTATED COMPARE VALUE TO COMPARE REGISTER
 0041 F245        184          JB7     ENCODE         ;TEST BIT 7 IF CHANGE DETECTED ENCODE CHANGE WORD
 0043 0469        185          JMP     CHFFUL         ;IF NO CHANGE IS DETECTED CHECK FOR FIFO FULL
 0045 FA          186 ENCODE:  MOV     A,R2           ;SCAN ROW SELECT TO ACCUMULATOR  0000XXXX
 0046 530F        187          ANL     A,#0FH         ;ROTATE ONLY SCAN VALUE
 0048 E7          188          RL      A              ;ROTATE LEFT                     000XXXX0
 0049 E7          189          RL      A              ;ROTATE LEFT                     00XXXX00
 004A E7          190          RL      A              ;ROTATE LEFT                     0XXXX000
 004B 4B          191          ORL     A,R3           ;ESTABLISH MATRIX COORDINANT     0XXXXXXX
                  192                                 ;(OR) COLUMN COUNTER VALUE WITH ACCUMULATOR
 004C AE          193          MOV     R6,A           ;SAVE COORDINANT IN CHANGE WORD REGISTER
 004D F0          194          MOV     A,@R0          ;COLUMN STATUS FROM MATRIX MAP TO ACCUMULATOR
 004E 5380        195          ANL     A,#80H         ;0 ALL BITS BUT BIT 7
 0050 4E          196          ORL     A,R6           ;(OR) SENSOR STATUS WITH COORDINATE FOR COMPLETED CHANGE WORD
 0051 AE          197          MOV     R6,A           ;SAVE CHANGE WORD                XXXXXXXX
                  198
                  199 $EJECT
```

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V3.0              PAGE    6

 LOC  OBJ          LINE        SOURCE STATEMENT

                    200 ;*********************************************************************
                    201 ;
                    202 ;                    FIFO-DBBOUT MANAGEMENT
                    203 ;
                    204 ;THE FOLLOWING CODE IS THE FIFO-DBBOUT MANAGEMENT SECTION OF THE
                    205 ;PROGRAM.   THIS SECTION TAKES AN ENCODED CHANGE WORD AND LOADS IT INTO
                    206 ;THE FIFO.  THE FIFO NOT EMPTY INTERRUPT IS THEN SET AND THE FIFO-IN
                    207 ;POINTER GETS UPDATED. A FIFO FULL CONDITION IS THEN CHECKED FOR AND
                    208 ;ROUTED ACCORDINGLY.   IF BOTH THE FIFO AND OBF HAVE CHANGE WORDS THE
                    209 ;PROGRAM LOCKS UP UNTIL THIS HAS CHANGED.   IF THE FIFO ISNT FULL COLUMN
                    210 ;COUNTER= 0, FIFO EMPTY AND OBF CONDITIONS ARE CHECKED.   THE FIFO-OUT
                    211 ;POINTER IS SET AND DBBOUT IS LOADED IF THE FIFO ISNT EMPTY AND OBF ISNT
                    212 ;SET.  IF THIS ISNT THE SITUATION, PROGRAM FLOW IS ROUTED BACK TO THE
                    213 ;THE SCAN AND COMPARE SECTION TO SCAN THE NEXT ROW.
                    214 ;
                    215 ;*********************************************************************
                    216 ;
0052 FC             217 LOADFF:  MOV    A,R4          ;FIFO INPUT ADDRESS TO ACCUMULATOR
0053 A9             218          MOV    R1,A          ;FIFO POINTER USED FOR INPUT
0054 FE             219          MOV    A,R6          ;CHANGE WORD TO ACCUMULATOR
0055 A1             220          MOV    @R1,A         ;LOAD FIFO AT FIFO INPUT ADDRESS
0056 2310           221 STATNE:  MOV    A,#10H        ;BIT 4 FOR FIFO NOT EMPTY
0058 90             222          MOV    STS,A         ;WRITE TO STATUS REGISTER, FIFO NOT EMPTY
0059 8A20           223 INTRH1:  ORL    P2,#20H       ;FIFO NOT EMPTY INTERRUPT PORT 2-5 HIGH
005B FA             224          MOV    A,R2          ;ROW SCAN SELECT TO ACCUMULATOR
005C 4320           225          ORL    A,#20H        ;SAVE INTERRUPT, NO CHANGE TO SCAN VALUE
005E AA             226          MOV    R2,A          ;ROW SCAN SELECT REGISTER
005F 232F           227 ADJFIN:  MOV    A,#FIFOTA     ;FIFO TOP ADDRESS TO ACCUMULATOR
0061 DC             228          XRL    A,R4          ;COMPARE WITH CURRENT FIFO INPUT ADDRESS
0062 C667           229          JZ     RSFFIN        ;IF THE SAME RESET FIFO INPUT REGISTER
0064 1C             230          INC    R4            ;NEXT FIFO INPUT ADDRESS
0065 0469           231          JMP    CHFFUL        ;CHECK FIFO FULL
0067 BC08           232 RSFFIN:  MOV    R4,#FIFOBA    ;RESET FIFO INPUT REGISTER,BOTTOM OF FIFO
0069 FC             233 CHFFUL:  MOV    A,R4          ;FIFO INPUT ADDRESS TO ACCUMULATOR
006A DD             234          XRL    A,R5          ;COMPARE INPUT WITH OUTPUT FIFO ADDRESS
006B 967D           235          JNZ    CHCNTR        ;IF NOT SAME CHECK COLUMN COUNTER VALUE
006D 866D           236 CHOBF1:  JOBF   CHOBF1        ;IF OBF IS 1 THEN CHECK OBF
006F 232F           237 ADJFOT:  MOV    A,#FIFOTA     ;FIFO TOP ADDRESS TO ACCUMULATOR
0071 DD             238          XRL    A,R5          ;COMPARE TOP TO OUTPUT FIFO ADDRESS
0072 C677           239          JZ     RSFFOT        ;IF THE SAME RESET FIFO OUTPUT REGISTER
0074 1D             240          INC    R5            ;NEXT FIFO OUTPUT ADDRESS
0075 0479           241          JMP    LOADDB        ;LOAD DBBOUT
0077 BD08           242 RSFFOT:  MOV    R5,#FIFOBA    ;RESET FIFO OUTPUT ADDRESS TO BOTTOM OF FIFO
0079 FD             243 LOADDB:  MOV    A,R5          ;OUTPUT FIFO ADDRESS TO ACCUMULATOR
007A A9             244          MOV    R1,A          ;FIFO POINTER USED FOR OUTPUT
007B F1             245          MOV    A,@R1         ;CHANGE WORD TO ACCUMULATOR
007C 02             246          OUT    DBB,A         ;CHANGE WORD TO DBBOUT
007D FB             247 CHCNTR:  MOV    A, R3         ;COLUMN COUNTER TO ACCUMULATOR
007E 963A           248          JNZ    RRLOOK        ;IF NOT 0 FINISH CHANGE WORD ENCODING
0080 2308           249 CHFFEM:  MOV    A,#FIFOBA     ;FIFO BOTTOM ADDRESS TO ACCUMULATOR
                    250
                    251 $EJECT
```

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V3 0              PAGE    7

 LOC  OBJ          LINE        SOURCE STATEMENT

 0082 DC            252          XRL     A,R4            ;COMPARE FIFO INPUT ADDRESS WITH FIFO BOTTOM ADD
 0083 C68C          253          JZ      ADJFEM          ;IF THE SAME, ADJUST TO CHECK FOR FIFO EMPTY
 0085 FC            254          MOV     A,R4            ;FIFO INPUT ADDRESS TO ACCUMULATOR
 0086 07            255          DEC     A               ;DECREMENT FIFO INPUT ADDRESS IN ACCUMULATOR
 0087 DD            256          XRL     A,R5            ;COMPARE INPUT TO OUTPUT FIFO ADDRESSES
 0088 C691          257          JZ      STATMT          ;IF SAME,WRITE STATUS REGISTER FOR FIFO EMPTY
 008A 049C          258          JMP     CHOBF2          ;CHECK OBF
 008C 232F          259 ADJFEM   MOV     A,#FIFOTA       ;FIFO TOP ADDRESS TO ACCUMULATOR
 008E DD            260          XRL     A,R5            ;COMPARE TOP TO OUTPUT FIFO ADDRESS
 008F 969C          261          JNZ     CHOBF2          ;IF NOT SAME THEN FIFO IS NOT EMPTY, CHECK OBF
 0091 2300          262 STATMT:  MOV     A,#00H          ;CLEAR BIT 0 FOR FIFO EMPTY
 0093 90            263          MOV     STS,A           ;WRITE TO STATUS REGISTER
 0094 9ADF          264 INTRLO:  ANL     P2,#0DFH        ;FIFO EMPTY, INTERRUPT PORT 2-5 LOW
 0096 FA            265          MOV     A,R2            ;SCAN ROW SELECT TO ACCUMULATOR
 0097 53DF          266          ANL     A,#0DFH         ;SAVE INTERRUPT, NO CHANGE TO SCAN VALUE
 0099 AA            267          MOV     R2,A            ;SCAN ROW SELECT REGISTER
 009A 041D          268          JMP     ADJREG          ;ADJUST REGISTERS
 009C 861D          269 CHOBF2:  JOBF    ADJREG          ;IF OBF=1 THEN ADJUST REGISTERS
 009E 046F          270          JMP     ADJFOT          ;ADJUST FIFO OUT ADDRESS TO LOAD DBBOUT
                    271
                    272          END

USER SYMBOLS
ADJFEM 008C    ADJFIN 005F    ADJFOT 006F    ADJREG 001D    CHCNTR 007D    CHFFEM 0080    CHFFUL 0069    CHOBF1 006D
CHOBF2 009C    DELAY2 0030    ENCODE 0045    FIFOBA 0008    FIFOTA 002F    FILLMX 000D    INITMX 0000    INTRH1 0059
INTRLO 0094    LOADDB 0079    LOADFF 0052    OBFINT 0018    RRLOOK 003A    RSETRG 0026    RSFFIN 0067    RSFFOT 0077
SCANMX 002C    SCANTM 000F    STATMT 0091    STATNE 0056

ASSEMBLY COMPLETE,   NO ERRORS
```

```
ISIS-II 8080/8085 MACRO ASSEMBLER, X108        MODULE   PAGE   1
8085A/UPI SENSOR MATRIX CONTROLLER

 LOC  OBJ        SEQ        SOURCE STATEMENT

                   1 ;
                   2 ;SUBROUTINE TO READ ALL CHANGES IN THE UPI AND BUILD A BUFFER
                   3 ;STARTING AT BUFSRT.  REG. B CONTAINS THE NUMBER OF CHANGES
                   4 ;UPON EXIT.  THE MAXIMUM NUMBER OF CHANGES IN ANY ONE CALL
                   5 ;IS 255.
                   6 ;
                   7 ;INPUTS: NOTHING
                   8 ;OUTPUTS: CHANGE WORD BUFFER AT BUFSRT
                   9 ;         CHANGE WORD COUNT IN REG. B
                  10 ;CALLS: NOTHING
                  11 ;
 4000              12          ORG     4000H
 00E5              13 STATUS   EQU     0E5H            ;UPI STATUS PORT
 00E4              14 DBBOUT   EQU     0E4H            ;UPI DBBOUT PORT
 0010              15 FIFO     EQU     10H             ;FIFO NOT EMPTY MASK
 0001              16 OBF      EQU     01H             ;OBF MASK
 4300              17 BUFSRT   EQU     4300H           ;BUFFER START LOCATION
                   18 ;
 4000 210043       19 START:   LXI     H,BUFSRT        ;INITIALIZE BUFFER POINTER
 4003 0600         20          MVI     B,00H           ;CLEAR CHANGE WORD COUNTER
 4005 DBE5         21 POLL1:   IN      STATUS          ;READ UPI STATUS
 4007 E611         22          ANI     FIFO OR OBF     ;TEST FIFO NOT EMPTY AND OBF
 4009 C8           23          RZ                      ;RETURN IF ZERO
 400A DBE5         24          IN      STATUS          ;READ UPI STATUS
 400C E601         25          ANI     OBF             ;TEST OBF FLAG
 400E CA0540       26          JZ      POLL1           ;WAIT IF NOT READY
 4011 DBE4         27          IN      DBBOUT          ;READ CHANGE WORD
 4013 77           28          MOV     M,A             ;LOAD BUFFER WITH CHANGE WORD
 4014 23           29          INX     H               ;INC BUFFER POINTER
 4015 04           30          INR     B               ;INC CHANGE WORD COUNTER
 4016 C8           31          RZ                      ;EXIT IF COUNTER = 256
 4017 C30540       32          JMP     POLL1           ;CHECK IF MORE CHANGE WORDS
                   33 ;
                   34 END
```

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V2.0
AP-41 COMBINATION I/O DEVICE

 LOC  OBJ        SEQ        SOURCE STATEMENT

                   1 $MOD42
                   2 ;*************************UNIOD*****************************************
                   3 ;
                   4 ;THIS UPI-41 PROGRAM IMPLEMENTS A FULL-DUPLEX UART WITH ON-CHIP
                   5 ;BAUD RATE GENERATION IN COMBINATION WITH AN 8-BIT PARALLEL I/O
                   6 ;PORT.  THE BAUD RATE IS SELECTABLE FROM 110 TO 1200 BAUD.  THE
                   7 ;PARALLEL I/O PORT IS PROGRAMMABLE FOR EITHER INPUT OR OUTPUT.
                   8 ;
                   9 ;INTERRUPT OUTPUTS ARE AVAILABLE FOR DATA AVAILABLE ON THE RECEIVER
                  10 ;AND PARALLEL INPUT.  THE STATUS REGISTER MUST BE READ TO DETERMINE
                  11 ;WHICH SOURCE CAUSED THE INTERRUPT.  THE FLAGS F0 AND F1 CODE THE
                  12 ;INTERRUPT SOURCE.  F0 AND F1 ALSO GIVE AN INDICATION OF COMMAND
                  13 ;ERRORS.
                  14 ;
                  15 ;*****************************************************************
                  16 ;
                  17 ;REGISTER DEFINITION
                  18 ;              RB0                    RB1
                  19 ;              ---                    ---
                  20 ;       0      NOT USED               NOT USED
                  21 ;       1      NOT USED               BAUD RATE CONSTANT
                  22 ;       2      NOT USED               TX TICK COUNTER
                  23 ;       3      RX STATUS (RXSTS)      TX SERIALIZER
                  24 ;       4      RX HOLDING             TX BUFFER
                  25 ;       5      RX TICK COUNTER        TX STATUS (TXSTS)
                  26 ;       6      RX DESERIALIZER        COMMAND STORE
                  27 ;       7      STATUS REG STORE       ACC. INTERRUPT SAVE
                  28 ;
                  29 ;*****************************************************************
                  30 ;
                  31 $EJECT
```

```
LOC  OBJ        SEQ        SOURCE STATEMENT

                 32 ;
                 33 ;*****************************************************************
                 34 ;
                 35 ;COMMANDS
                 36 ;
                 37 ;       CONFIGURE: 0 0 0 A B C D P
                 38 ;                                          A - 1200 BAUD SELECT
                 39 ;                                          B - 600 BAUD SELECT
                 40 ;                                          C - 300 BAUD SELECT
                 41 ;                                          D - 110 BAUD SELECT
                 42 ;                                          E - PARALLEL I/O DIRECTION
                 43 ;                                                  0 - INPUT
                 44 ;                                                  1 - OUTPUT
                 45 ;
                 46 ;       I/O:          1 0 0 0 0 0 0 0    (PERFORM I/O OPERATION)
                 47 ;       RESET ERROR:1 1 0 0 0 0 0 0      (RESET RX ERROR IN STATUS)
                 48 ;
                 49 ;*****************************************************************
                 50 ;
                 51 ;STATUS REGISTER DEFINITION
                 52 ;
                 53 ;       BIT            DEFINITION
                 54 ;       ---            ----------
                 55 ;        0             OBF - DATA AVAILABLE
                 56 ;        1             IBF - BUSY
                 57 ;        2             F0
                 58 ;        3             F1
                 59 ;        4             NOT USED
                 60 ;        5             TXINT - TX INTERRUPT
                 61 ;        6             FRAMING ERROR
                 62 ;        7             OVERRUN ERROR
                 63 ;
                 64 ;       F0      F1     OPERATION
                 65 ;       --      --     ---------
                 66 ;       0       0      X
                 67 ;       0       1      PARALLEL I/O DATA AVAILABLE
                 68 ;       1       0      SERIAL I/O DATA AVAILABLE
                 69 ;       1       1      COMMAND ERROR
                 70 ;
                 71 ;*****************************************************************
                 72 ;
                 73 $EJECT
```

```
LOC  OBJ        SEQ        SOURCE STATEMENT

                 74 ;
                 75 ;*****************************************************************
                 76 ;
                 77 ;STATUS REGISTER DEFINITIONS
                 78 ;
                 79 ;                        RXSTS                    TXSTS
                 80 ;                        ----                     ----
                 81 ;                 0      RX FLAG - SPACE          TX FLAG - TRANSMITTING CHR
                 82 ;                 1      START FLAG - GOOD START  REQUEST BYTE - CHR IN BUFFER
                 83 ;                 2      BYTE FINISHED            TX PIPELINED DATA BIT
                 84 ;                 3      DATA READY               START BIT FLAG
                 85 ;                 4      FRAMING ERROR            NOT USED
                 86 ;                 5      OVERRUN ERROR            NOT USED
                 87 ;                 6      IO DIRECTION             NOT USED
                 88 ;                 7      IO FLAG                  NOT USED
                 89 ;
                 90 ;*****************************************************************
                 91 ;
                 92 ;PORT 2 DEFINITIONS
                 93 ;
                 94 ;        BIT            DEFINITION
                 95 ;        ---            ----------
                 96 ;         0             TX DATA
                 97 ;         1             NOT USED
                 98 ;         2             NOT USED
                 99 ;         3             TX INTERRUPT
                100 ;         4             OBF INTERRUPT (RX OR I/O DATA AVAILABLE)
                101 ;         5             NOT USED
                102 ;         6             NOT USED (TICK SAMPLE)
                103 ;         7             NOT USED
                104 ;
                105 ;*****************************************************************
                106 ;
                107 ;MISC.
                108 ;
                109 ;        RX DATA        T0 INPUT
                110 ;        EXT CLOCK      T1 INPUT 76.8KHZ (1.2288MHZ/16)
                111 ;
                112 ;*****************************************************************
                113 ;
                114 $EJECT
```

```
LOC  OBJ        SEQ        SOURCE STATEMENT

                115 ;
                116 ;*****************************************************************
                117 ;
                118 ;SYSTEM EQUATES:
                119 ;
0001            120 RXFLG   EQU     01H             ;RECEIVE FLAG IN RXSTS
0002            121 SRTFLG  EQU     02H             ;START BIT FLAG IN RXSTS
0004            122 BFFLG   EQU     04H             ;BYTE FINISHED FLAG IN RXSTS
0008            123 DATRDY  EQU     08H             ;DATA READY FLAG IN RXSTS
0010            124 FRAMER  EQU     10H             ;FRAMING ERROR FLAG IN RXSTS
0020            125 OVRUN   EQU     20H             ;OVERRUN ERROR FLAG IN RXSTS
0040            126 IODIR   EQU     40H             ;I/O DIRECTION FLAG IN RXSTS
0080            127 IOFLG   EQU     80H             ;I/O REQUEST FLAG IN RXSTS
0001            128 TXFLG   EQU     01H             ;TX FLAG IN TXSTS
0002            129 REQFLG  EQU     02H             ;REQUEST BYTE FLAG IN TXSTS
0040            130 TICOUT  EQU     40H             ;TICK SAMPLE BIT IN PORT 2
0080            131 RXINTL  EQU     80H             ;RX DESERIALIZER INITIALIZATION
0004            132 TICSRT  EQU     04H             ;TICK INITIALIZATION
007F            133 ASCMSK  EQU     7FH             ;ASCII MASK
0003            134 TXTIC   EQU     03H             ;TX TICK MOD MASK
0028            135 TXEND   EQU     40D             ;TICK COUNT AT END OF TX CHARACTER
0024            136 STPEND  EQU     36D             ;TICK COUNT AT END OF TX DATA
0004            137 MARK    EQU     04H             ;MARK OUTPUT
00FB            138 SPACE   EQU     0FBH            ;SPACE OUTPUT
0000            139 ZERO    EQU     00H             ;GENERAL CLEAR
0008            140 TXINT   EQU     08H             ;TX INTERRUPT OUTPUT IN PORT 2
0020            141 TXBIT   EQU     20H             ;TX INTERRUPT BIT IN STATUS
0020            142 TIMCON  EQU     32D             ;TIMER CONSTANT RAM LOCATION
003F            143 RSTERR  EQU     3FH             ;RESET ERROR MASK FOR STATUS
0040            144 FESTS   EQU     40H             ;FRAMING ERROR BIT IN STATUS
0080            145 OVSTS   EQU     80H             ;OVERRUN ERROR BIT IN STATUS
0001            146 MKOUT   EQU     01H             ;MARK OUTPUT TO PORT
00FE            147 SPOUT   EQU     0FEH            ;SPACE OUTPUT TO PORT
0008            148 SBIT    EQU     08H             ;TX START BIT FLAG
0003            149 RXSTS   EQU     R3              ;RX STATUS REGISTER
0005            150 TXSTS   EQU     R5              ;TX STATUS REGISTER
                151 ;
                152 $EJECT
```

```
LOC  OBJ        SEQ        SOURCE STATEMENT

                153 ;*****************************************************************
                154 ;
                155 ;RESET VECTOR LOCATION
                156 ;
                157 ;*****************************************************************
                158 ;
0000            159        ORG     0000H
                160 ;
0000 C5         161 RESET:  SEL     RB0            ;GET INTO RB0 AT RESET
0001 4400       162        JMP     INIT           ;GO TO INITIALIZATION
                163 ;
                164 ;*****************************************************************
                165 ;
                166 ;TIMER INTERRUPT LOCATION - TIMER IS SET TO 4 TIMES THE BAUD RATE.  THE
                167 ;RECEIVER AND TRANSMITTER ARE SERVICED EVERY FOUR TIMER TICKS.  SOFTWARE
                168 ;DELAY LOOP IS USED FOR TIMING FINE-TUNING.  RB1 R1 POINTS AT DELAY
                169 ;CONSTANT AT INTERRUPT.  R1-1 POINTS AT TIMER CONSTANT.
                170 ;
                171 ;*****************************************************************
                172 ;
0007            173        ORG     0007H
                174 ;
0007 D5         175 TIMINT: SEL     RB1            ;INTERRUPT PROCESSING IN RB1
0008 AF         176        MOV     R7,A           ;SAVE ACCUMULATOR IN R7
0009 F9         177        MOV     A,R1           ;GET TIMER CONSTANT
000A 00         178        NOP                    ;DELAY TO GET INTO T1 HIGH
000B 560B       179 INT1:   JT1     INT1           ;WAIT UNTIL T1 IS LOW
000D 62         180        MOV     T,A            ;THEN LOAD COUNTER
                181 ;
                182 ;TICK SAMPLE OUTPUT
                183 ;
000E 9ABF       184        ANL     P2,#NOT TICOUT
0010 8A40       185        ORL     P2,#TICOUT
                186 ;
                187 ;*****************************************************************
                188 ;
                189 ;TRANSMITTER OUTPUT - TIME CRITICAL TASKS DONE FIRST.  DATA BIT OUTPUT
                190 ;PIPELINED IN TXSTS BIT 2 IS OUTPUT NOW.
                191 ;
                192 ;*****************************************************************
                193 ;
0012 FD         194 TXOUT:  MOV     A,TXSTS        ;GET TX STATUS
0013 5219       195        JB2     MOUT           ;TEST PIPELINED DATA
0015 9AFE       196        ANL     P2,#SPOUT      ;OUTPUT SPACE IF RESET
0017 041B       197        JMP     RCV            ;DO RECEIVER
0019 8A01       198 MOUT:   ORL     P2,#MKOUT      ;OUTPUT MARK IS SET
                199 ;
                200 ;*****************************************************************
                201 ;
                202 ;START OF RECEIVER FLOW -  RXSTS REGISTER
                203 ;HOLDS RECEIVER STATUS.
                204 ;
                205 ;*****************************************************************
                206 ;
001B C5         207 RCV:    SEL     RB0            ;SWITCH TO RX BANK
```

```
LOC  OBJ        SEQ        SOURCE STATEMENT

001C FB         208        MOV    A,RXSTS       ;GET RXSTS
001D 1226       209        JB0    RCV1          ;TEST RECEIVE FLAG
                210                             ;0 - NO CHR BEING RECEIVED
                211                             ;1 - POSSIBLE START BIT, DO TEST
001F 3668       212        JT0    XMIT          ;TEST RXD INPUT
                213                             ;0 - SPACE, SET RX FLAG
                214                             ;1 - MARK, GO CHECK XMIT
0021 4301       215        ORL    A,#RXFLG      ;SPACE - SET RX FLAG
0023 AB         216        MOV    RXSTS,A       ;RESTORE RXSTS
0024 0468       217        JMP    XMIT          ;GO HANDLE XMTR
                218 ;
                219 ;START BIT TEST
                220 ;
0026 3238       221 RCV1:  JB1    RCV3          ;FIRST TEST START BIT FLAG
0028 3633       222        JT0    RCV2          ;TEST RXD INPUT
                223                             ;0 - SPACE, GOOD START BIT
                224                             ;1 - MARK, BAD START BIT, IGNORE
002A 4302       225        ORL    A,#SRTFLG     ;GOOD START - SET START BIT FLAG
002C AB         226        MOV    RXSTS,A       ;RESTORE RXSTS
002D BE80       227        MOV    R6,#RXINTL    ;SETUP RX DESERIALIZER
002F BD04       228        MOV    R5,#TICSRT    ;LOAD RX TICK COUNTER
0031 0468       229        JMP    XMIT          ;GO HANDLE XMTR
                230 ;
                231 ;BAD START BIT - RESET FLAGS
                232 ;
0033 53FE       233 RCV2:  ANL    A,#NOT RXFLG  ;RESET RECEIVE FLAG
0035 AB         234        MOV    RXSTS,A       ;RESTORE RXSTS
0036 0468       235        JMP    XMIT          ;GO HANDLE XMTR
                236 ;
                237 ;IN MIDDLE OF CHR - SAMPLE EVERY 4 TIMER TICKS
                238 ;
0038 ED68       239 RCV3:  DJNZ   R5,XMIT       ;WAIT UNTIL 4TH TICK
003A BD04       240        MOV    R5,#TICSRT    ;RELOAD RX TICK COUNTER
003C 524D       241        JB2    RCV5          ;TEST BYTE FINISHED FLAG
                242                             ;0 - MIDDLE OF CHR, CONTINUE
                243                             ;1 - DONE WITH STOP BITS
003E 97         244        CLR    C             ;CLEAR CARRY BEFORE ROTATE
003F 2642       245        JNT0   RCV4          ;TEST RXD INPUT
0041 A7         246        CPL    C             ;RXD IS MARK, SET CARRY
0042 FE         247 RCV4:  MOV    A,R6          ;GET DESERIALIZER
0043 67         248        RRC    A             ;ROTATE IN NEW BIT
0044 AE         249        MOV    R6,A          ;RESTORE DESERIALIZER
0045 E668       250        JNC    XMIT          ;TEST CARRY AFTER ROTATE
                251                             ;0 - MIDDLE OF CHR
                252                             ;1 - STOP BIT COMING NEXT
0047 FB         253        MOV    A,RXSTS       ;GET RXSTS
0048 4304       254        ORL    A,#BFFLG      ;SET BYTE FINISHED FLAG
004A AB         255        MOV    RXSTS,A       ;RESTORE RXSTS
004B 0468       256        JMP    XMIT          ;GO HANDLE XMTR
                257 ;
                258 ;BYTE FINISHED - DO STOP BIT TEST
                259 ;
004D 2660       260 RCV5:  JNT0   RCV8          ;TEST RXD INPUT
                261                             ;0 - SPACE, INVALID STOP BIT
                262                             ;1 - MARK, VALID STOP BIT
```

```
LOC  OBJ        SEQ        SOURCE STATEMENT

004F 53EF                  ANL     A,#NOT FRAMER   ;NO FRAMING ERROR, RESET FLAG
                  ;
                  ;OVERRUN TEST - IF RX DATA READY STILL SET, OVERRUN ERROR
                  ;
0051 7264         RCV6:    JB3     RCV9            ;IF DATA READY STILL SET, ERROR
0053 53DF                  ANL     A,#NOT OVRUN    ;NO OVERRUN, RESET FLAG
                  ;
                  ;CLEAN UP RXSTS AT CHR COMPLETE
                  ;
0055 4308         RCV7:    ORL     A,#DATRDY       ;SET DATA READY
0057 53F8                  ANL     A,#NOT (RXFLG OR SRTFLG OR BFFLG)       ;RESET OTHER FLAGS
0059 AB                    MOV     RXSTS,A         ;RESTORE RXSTS
005A FE                    MOV     A,R6            ;GET DESERIALIZER REG
005B 537F                  ANL     A,#ASCMSK       ;MAKE IT 7 BITS
005D AC                    MOV     R4,A            ;PUT DATA INTO HOLDING REG
005E 0468                  JMP     XMIT            ;GO HANDLE XMTR
                  ;
                  ;BAD STOP - SET FRAMING ERROR FLAG
                  ;
0060 4310         RCV8:    ORL     A,#FRAMER       ;SET FRAMING ERROR FLAG
0062 0451                  JMP     RCV6            ;CONTINUE
                  ;
                  ;OVERRUN ERROR - SET OVERRUN FLAG
                  ;
0064 4320         RCV9:    ORL     A,#OVRUN        ;SET OVERRUN FLAG
0066 0455                  JMP     RCV7            ;CONTINUE
                  ;
                  ;*****************************************************************
                  ;
                  ;START OF TRANSMITTER FLOW - TRANSMITTER IS SERVICED EVERY 4 TICKS.
                  ;THE TX TICK COUNTER SERVES AS THE TX BIT COUNTER.  TRANSMITTER STATUS
                  ;IS HELD IN THE TXSTS REGISTER.
                  ;
                  ;*****************************************************************
                  ;
0068 D5           XMIT:    SEL     RB1             ;BE SURE WE'RE IN RB1
0069 FD                    MOV     A,TXSTS         ;GET TX STATUS
006A 72B3                  JB3     SRTBIT          ;THIS IS START OF START BIT
006C 1A                    INC     R2              ;INC TX TICK COUNTER
006D 2303                  MOV     A,#TXTIC        ;TEST TICK COUNTER MOD 4
006F 5A                    ANL     A,R2
0070 96B0                  JNZ     RETURN          ;IF NON-ZERO, MIDDLE OF BIT
0072 FD                    MOV     A,TXSTS         ;ZERO, GET TXSTS
0073 37                    CPL     A               ;COMPLEMENT FOR 0 TEST
0074 129C                  JB0     XMT4            ;TEST TX FLAG
                                                   ;0 - NOT TXING, CHECK FOR NEW CHR
                                                   ;1 - CURRENTLY IN CHR
0076 2328                  MOV     A,#TXEND        ;CHECK FOR END OF DATA AND STOP
0078 DA                    XRL     A,R2            ;XOR WITH CURRENT TICK COUNT
0079 9681                  JNZ     XMT1            ;NOT DONE, CONTINUE
007B FD                    MOV     A,TXSTS         ;DONE, GET TXSTS
007C 53FE                  ANL     A,#NOT TXFLG    ;RESET TX FLAG
007E AD                    MOV     TXSTS,A         ;RESTORE TXSTS
007F 04B0                  JMP     RETURN          ;GO EXIT
                  ;
```

```
LOC  OBJ        SEQ        SOURCE STATEMENT

                318 ;CHECK IF IT'S TIME FOR STOP BIT
                319 ;
0081 2324       320 XMT1:   MOV     A,#STPEND       ;CHECK FOR STOP BIT TIME
0083 DA         321         XRL     A,R2            ;COMPARE WITH TICK COUNTER
0084 968C       322         JNZ     XMT2            ;NOT TIME, DO NEXT BIT
                323 ;
                324 ;TRANSMIT STOP BIT
                325 ;
0086 FD         326         MOV     A,TXSTS         ;GET TX STATUS
0087 4304       327         ORL     A,#MARK         ;SETUP PIPELINED STOP BIT
0089 AD         328         MOV     TXSTS,A         ;RESTORE TX STATUS
008A 04B0       329         JMP     RETURN          ;RETURN
                330 ;
                331 ;IN MIDDLE OF CHR - TRANSMIT NEXT BIT
                332 ;
008C FB         333 XMT2:   MOV     A,R3            ;GET TX SERIALIZER
008D 67         334         RRC     A               ;ROTATE NEXT BIT INTO CARRY
008E AB         335         MOV     R3,A            ;RESTORE SERIALIZER
008F FD         336         MOV     A,TXSTS         ;GET TX STATUS FOR PIPELINED DATA
0090 F697       337         JC      XMT3            ;OUTPUT A MARK IF 1
0092 53FB       338         ANL     A,#SPACE        ;RESET TXDATA BIT
0094 AD         339         MOV     TXSTS,A         ;RESTORE TX STATUS
0095 04B0       340         JMP     RETURN          ;GO EXIT
0097 4304       341 XMT3:   ORL     A,#MARK         ;SET TXDATA BIT
0099 AD         342         MOV     TXSTS,A         ;RESTORE TX STATUS
009A 04B0       343         JMP     RETURN          ;GO EXIT
                344 ;
                345 ;TEST REQUEST FLAG SINCE NOT CURRENTLY TRANSMITTING
                346 ;
009C 32A8       347 XMT4:   JB1     XMT5            ;TEST TX REQUEST FLAG
                348                                 ;0 - NO CHR WAITING IN BUFFER
                349                                 ;1 - CHR WAITING IN BUFFER
009E FC         350         MOV     A,R4            ;CHR WAITING, GET IT FROM HOLDING
009F AB         351         MOV     R3,A            ;PUT IN SERIALIZER
00A0 FD         352         MOV     A,TXSTS         ;GET TXSTS
00A1 53FD       353         ANL     A,#NOT REQFLG   ;RESET REQUEST FLAG
00A3 4309       354         ORL     A,#TXFLG OR SBIT ;SET TX AND START BIT FLAGS
00A5 53FB       355         ANL     A,#SPACE        ;SETUP TXDATA FOR START BIT
00A7 AD         356         MOV     TXSTS,A         ;RESTORE TXSTS
                357 ;
                358 ;TX BUFFER EMPTY - SET TXINT PIN AND BIT
                359 ;
00A8 8A08       360 XMT5:   ORL     P2,#TXINT       ;SET TXINT PIN
00AA C5         361         SEL     RB0             ;SWITCH FOR STS
00AB FF         362         MOV     A,R7            ;GET STS
00AC 4320       363         ORL     A,#TXBIT        ;SET TXINT BIT
00AE AF         364         MOV     R7,A            ;RESTORE STS
00AF 90         365         MOV     STS,A           ;LOAD STATUS
                366 ;
                367 ;*****************************************************************
                368 ;
                369 ;EXIT FOR TIMER INTERRUPT ROUTINE POINT
                370 ;
                371 ;*****************************************************************
                372 ;
00B0 D5         373 RETURN: SEL     RB1             ;MAKE SURE WE'RE IN RB1
00B1 FF         374         MOV     A,R7            ;RESTORE A
00B2 93         375         RETR                    ;RETURN WITH RESTORE
                376 ;
                377 ;*****************************************************************
                378 ;
                379 ;GET HERE IF INTERRUPT IS FIRST FOR START BIT - CLEAR START BIT FLAG IN
                380 ;TXSTS AND SETUP TX TICK COUNTER.
                381 ;
                382 ;*****************************************************************
                383 ;
00B3 53F7       384 SRTBIT: ANL     A,#NOT SBIT     ;RESET START BIT FLAG IN TXSTS
00B5 AD         385         MOV     TXSTS,A         ;RESTORE TX STATUS
00B6 BA01       386         MOV     R2,#01H         ;INITIALIZE TX TICK COUNTER
00B8 04B0       387         JMP     RETURN          ;RETURN
                388 ;
                389 $EJECT
```

```
LOC  OBJ        SEQ        SOURCE STATEMENT

                390 ;
                391 ;*****************************************************************
                392 ;
                393 ;COMMAND RECOGNIZER - GET HERE FROM I8F WRITE WITH F1 SET.  COMMAND
                394 ;IS STORED IN R6.  BAUD RATE SELECTION BITS ARE EVALUATED RIGHT TO LEFT.
                395 ;THE FIRST SET BIT FOUND DETERMINES THE BAUD RATE.  IF AN INVALID COMMAND
                396 ;IS DETECTED, BOTH F1 AND F0 ARE SET AND NO ACTION IS TAKEN.
                397 ;THE TIMER BAUD RATE CONSTANT IS SET TO TWO COUNTS LESS THAN THE DESIRED
                398 ;NUMBER.
                399 ;
                400 ;*****************************************************************
                401 ;
0100            402        ORG     0100H
                403 ;
0100 D5         404 CMD:   SEL     RB1          ;SELECT RB1
0101 22         405        IN      A,DBB        ;READ COMMAND
0102 AE         406        MOV     R6,A         ;SAVE COMMAND IN R6
0103 F227       407        JB7     IOER         ;IF BIT 7 SET, IO OPERATION
0105 53E0       408        ANL     A,#0E0H      ;TEST TOP 3 BITS
0107 963A       409        JNZ     ERROR        ;IF NON-ZERO, ERROR
0109 C5         410        SEL     RB0          ;IO FLAG IN RB0
010A 1221       411        JB0     CMD2         ;IF BIT 0=1, OUTPUT PORT
010C 89FF       412        ORL     P1,#0FFH     ;INPUT PORT, SET ALL HIGH
010E FB         413        MOV     A,RXSTS      ;GET RXSTS
010F 53BF       414        ANL     A,#NOT IODIR ;RESET IO DIRECTION FLAG
0111 AB         415        MOV     RXSTS,A      ;RESTORE RXSTS
0112 D5         416 CMD1:  SEL     RB1          ;BAUD RATE CONSTANTS IN RB1
0113 B920       417        MOV     R1,#TIMCON   ;POINT AT TIMER CONSTANT LOCATION
0115 FE         418        MOV     A,R6         ;GET COMMAND
0116 323E       419        JB1     B110         ;110 BAUD SELECTED
0118 5242       420        JB2     B300         ;300 BAUD SELECTED
011A 7246       421        JB3     B600         ;600 BAUD SELECTED
011C 924A       422        JB4     B1200        ;1200 BAUD SELECTED
011E B5         423        CPL     F1           ;RESET F1
011F 4414       424        JMP     MNLP1        ;DONE, JUMP BACK TO MAIN LOOP
                425 ;
                426 ;PORT IS SELECTED AS OUTPUT PORT - SET IO DIRECTION FLAG
                427 ;
0121 FB         428 CMD2:  MOV     A,RXSTS      ;GET RXSTS
0122 4340       429        ORL     A,#IODIR     ;SET IO DIRECTION FLAG
0124 AB         430        MOV     RXSTS,A      ;RESTORE RXSTS
0125 2412       431        JMP     CMD1         ;CONTINUE
                432 ;
                433 ;HERE WITH EITHER IO OR RESET ERROR COMMAND
                434 ;
0127 D231       435 IOER:  JB6     ERRST        ;IF BIT 6 SET, RESET ERROR FLAGS
0129 C5         436        SEL     RB0          ;IO FLAG IN RXSTS
012A FB         437        MOV     A,RXSTS      ;GET RXSTS
012B 4380       438        ORL     A,#IOFLG     ;SET IO FLAG
012D AB         439        MOV     RXSTS,A      ;RESTORE RXSTS
012E B5         440        CPL     F1           ;RESET F1
012F 4414       441        JMP     MNLP1        ;DONE, JUMP BACK TO MAIN LOOP
                442 ;
                443 ;RESET ERROR COMMAND
                444 ;
```

```
LOC  OBJ        SEQ        SOURCE STATEMENT

0131 C5          445 ERRST:  SEL     RB0            ;STS IN RB0
0132 FF          446         MOV     A,R7           ;GET STS
0133 533F        447         ANL     A,#RSTERR      ;RESET ERROR FLAGS
0135 AF          448         MOV     R7,A           ;RESTORE STS
0136 90          449         MOV     STS,A          ;LOAD STATUS
0137 B5          450         CPL     F1             ;RESET F1
0138 4414        451         JMP     MNLP1          ;DONE, BACK TO MAIN LOOP
                 452 ;
                 453 ;COMMAND ERROR - SET BOTH F1 AND F0
                 454 ;
013A 85          455 ERROR:  CLR     F0             ;SET F0
013B 95          456         CPL     F0
013C 4414        457         JMP     MNLP1          ;DONE, BACK TO MAIN LOOP
                 458 ;
                 459 ;110 BAUD CONSTANTS
                 460 ;
013E B954        461 B110:   MOV     R1,#-(174D-2D) ;LOAD 110 BAUD CONSTANT
0140 244C        462         JMP     STTIMR         ;GO START TIMER
                 463 ;
                 464 ;300 BAUD CONSTANTS
                 465 ;
0142 B9C2        466 B300:   MOV     R1,#-(64D-2D)  ;LOAD 300 BAUD CONSTANT
0144 244C        467         JMP     STTIMR         ;GO START COUNTER
                 468 ;
                 469 ;600 BAUD CONSTANTS
                 470 ;
0146 B9E2        471 B600:   MOV     R1,#-(32D-2D)  ;LOAD 600 BAUD CONSTANT
0148 244C        472         JMP     STTIMR         ;GO START COUNTER
                 473 ;
                 474 ;1200 BAUD CONSTANTS
                 475 ;
014A B9F2        476 B1200:  MOV     R1,#-(16D-2D)  ;LOAD 1200 BAUD CONSTANT
                 477 ;
                 478 ;START COUNTER
                 479 ;
014C F9          480 STTIMR: MOV     A,R1           ;GET COUNTER CONSTANT
014D 62          481         MOV     T,A            ;LOAD COUNTER
014E 45          482         STRT    CNT            ;START COUNTER
014F 25          483         EN      TCNTI          ;ENABLE TIMER INTERRUPTS
0150 B5          484         CPL     F1             ;RESET F1
0151 4414        485         JMP     MNLP1          ;DONE, BACK TO MAIN LOOP
                 486 ;
                 487 $EJECT
```

```
LOC  OBJ        SEQ        SOURCE STATEMENT

                488 ;
                489 ;*****************************************************************
                490 ;
                491 ;DATA ROUTINE - GET HERE WITH IBF WRITE WITH F1 RESET.  THIS ROUTINE
                492 ;FIRST TESTS IF THE I/O FLAG IS SET IN THE RXSTS REGISTER.  IF SO, THE DATA
                493 ;IS FOR THE OUTPUT PORT.  OTHERWISE, THE DATA IS FOR THE TRANSMITTER AND
                494 ;IS PLACED IN THE TX BUFFER REGISTER.  THE TXINT BIT AND PIN ARE RESET.
                495 ;
                496 ;*****************************************************************
                497 ;
0153 C5         498 DATA:   SEL     RB0            ;DATA HANDLED MOSTLY IN RB0
0154 FB         499         MOV     A,RXSTS        ;GET RXSTS
0155 F267       500         JB7     IODATA         ;IF IO FLAG SET, DATA IN FOR I/O
0157 FF         501         MOV     A,R7           ;GET STS
0158 53DF       502         ANL     A,#NOT TXBIT   ;RESET TXINT BIT IN STS
015A AF         503         MOV     R7,A           ;RESTORE STS
015B 90         504         MOV     STS,A          ;LOAD STATUS
015C 9AF7       505         ANL     P2,#NOT TXINT  ;RESET TXINT PIN
015E D5         506         SEL     RB1            ;TXSTS IN RB1
015F 22         507         IN      A,DBB          ;READ DATA
0160 AC         508         MOV     R4,A           ;PUT DATA IN TX BUFFER
0161 FD         509         MOV     A,TXSTS        ;GET TXSTS
0162 4382       510         ORL     A,#REQFLG      ;SET REQUEST FLAG IN TXSTS
0164 AD         511         MOV     TXSTS,A        ;RESTORE TXSTS
0165 4414       512         JMP     MNLP1          ;BACK TO MAIN LOOP
                513 ;
                514 ;IO DATA ROUTINE
                515 ;
0167 537F       516 IODATA: ANL     A,#NOT IOFLG   ;RESET IO FLAG
0169 AB         517         MOV     RXSTS,A        ;RESTORE RXSTS
016A 22         518         IN      A,DBB          ;READ IO DATA FROM DBBIN
016B 39         519         OUTL    P1,A           ;OUTPUT TO PORT 1
016C 4414       520         JMP     MNLP1          ;DONE, BACK TO MAIN LOOP
                521 ;
                522 $EJECT
```

```
LOC  OBJ        SEQ        SOURCE STATEMENT

                523 ;
                524 ;*****************************************************************
                525 ;
                526 ;INITIALIZATION - GET HERE AT RESET.  THIS ROUTINE RESETS THE INTERRUPT
                527 ;OUTPUTS AND ENABLES THEM, AND CLEARS THE APPROPRIATE STATUS AND DATA
                528 ;REGISTERS.
                529 ;
                530 ;*****************************************************************
                531 ;
0200            532          ORG     0200H
                533 ;
0200 9AF7       534 INIT:    ANL     P2,#0F7H          ;RESET TXINT PIN
0202 F5         535          EN      FLAGS             ;ENABLE INTERRUPTS OUTPUT
0203 2300       536          MOV     A,#ZERO           ;CLEAR A
0205 AB         537          MOV     RXSTS,A           ;CLEAR RXSTS
0206 AD         538          MOV     R5,A              ;CLEAR RX TICK COUNTER
0207 AF         539          MOV     R7,A              ;CLEAR STS
0208 D5         540          SEL     RB1               ;SWITCH BANKS
0209 AE         541          MOV     R6,A              ;CLEAR CONFIGURE STORE
020A B004       542          MOV     TXSTS,#MARK       ;SETUP PIPELINED TX DATA
                543 ;
                544 ;*****************************************************************
                545 ;
                546 ;MAIN LOOP - IBF AND OBF ARE HANDLED IN THIS LOOP.  IF IBF=1, THE
                547 ;APPROPRIATE COMMAND OR DATA ROUTINE IS ACCESSED.  IF IBF=0, THEN OBF
                548 ;IS TESTED.  IF OBF=1, IBF IS TESTED AGAIN.  AS SOON AS OBF=0, RXSTS
                549 ;IS EXAMINED TO SEE IF DATA IS WAITING FOR OUTPUT.  WHEN RX DATA
                550 ;READY IS SET, F0 IS SET AND F1 IS CLEARED, AND THE DATA IS TRANSFERRED
                551 ;FROM THE RX HOLDING REGISTER INTO DBBOUT AFTER TESTING FOR ERROR
                552 ;FLAGS.  ANY ERROR FLAGS SET ARE TRANSFERRED TO THE STATUS REGISTER.
                553 ;IF THE I/O FLAG IS SET, THE PORT IS READ AND THE DATA TRANSFERRED TO
                554 ;DBBOUT.
                555 ;
                556 ;*****************************************************************
                557 ;
020C D614       558 MNLOOP:  JNIBF   MNLP1             ;IF IBF=0, TEST OBF
020E 7612       559          JF1     CMDJ1             ;IBF=1, TEST F1 FOR COMMAND
0210 2453       560          JMP     DATA              ;F1=0, JUMP TO DATA ROUTINE
0212 2400       561 CMDJ1:   JMP     CMD               ;OUT-OF-PAGE COMMAND JUMP
0214 860C       562 MNLP1:   JOBF    MNLOOP            ;WAIT UNTIL DBBOUT IS FREE
0216 C5         563          SEL     RB0               ;RXSTS IN RB0
0217 FB         564          MOV     A,RXSTS           ;GET RXSTS
0218 721E       565          JB3     RXRDY             ;TEST RX DATA READY FLAG
021A F23C       566          JB7     IOFLAG            ;TEST IO FLAG
021C 440C       567          JMP     MNLOOP            ;LOOP
                568 ;
                569 ;RX DATA READY - TRANSFER TO DBBOUT
                570 ;
021E 85         571 RXRDY:   CLR     F0                ;SET F0
021F 95         572          CPL     F0
0220 A5         573          CLR     F1                ;RESET F1
0221 922E       574          JB4     RXF               ;CHECK FRAMING ERROR FLAG
0223 FB         575 RXRDY1:  MOV     A,RXSTS           ;GET RXSTS
0224 B235       576          JB5     RXO               ;CHECK FOR OVERRUN ERROR
0226 FB         577 RXRDY2:  MOV     A,RXSTS           ;GET RXSTS AGAIN
```

```
LOC  OBJ        SEQ        SOURCE STATEMENT

0227 53C7        578        ANL     A,#NOT (DATRDY OR FRAMER OR OVRUN)     ;RESET SOME FLAGS
0229 AB          579        MOV     RXSTS,A        ;RESTORE RXSTS
022A FC          580        MOV     A,R4           ;GET DATA FROM HOLDING REG
022B 02          581        OUT     DBB,A          ;PUT IN DBBOUT
022C 440C        582        JMP     MNLOOP         ;LOOP
                 583 ;
                 584 ;FRAMING ERROR FLAG SET
                 585 ;
022E FF          586 RXF:   MOV     A,R7           ;GET STS
022F 4340        587        ORL     A,#FESTS       ;SET FRAMING ERROR FLAG
0231 AF          588        MOV     R7,A           ;RESTORE STS
0232 90          589        MOV     STS,A          ;LOAD STATUS
0233 4423        590        JMP     RXRDY1         ;CONTINUE
                 591 ;
                 592 ;OVERRUN ERROR FLAG SET
                 593 ;
0235 FF          594 RXO:   MOV     A,R7           ;GET STS
0236 4380        595        ORL     A,#OVSTS       ;SET OVERRUN ERROR FLAG
0238 AF          596        MOV     R7,A           ;RESTORE STS
0239 90          597        MOV     STS,A          ;LOAD STATUS
023A 4426        598        JMP     RXRDY2         ;CONTINUE
                 599 ;
                 600 ;IO FLAG SET - TEST DIRECTION
                 601 ;
023C FB          602 IOFLAG: MOV    A,RXSTS        ;GET RXSTS
023D D20C        603        JB6     MNLOOP         ;PORT IS OUTPUT - NO ACTION
023F 85          604        CLR     F0             ;RESET F0
0240 A5          605        CLR     F1             ;SET F1
0241 B5          606        CPL     F1
0242 537F        607        ANL     A,#NOT IOFLG   ;RESET IO FLAG
0244 AB          608        MOV     RXSTS,A        ;RESTORE RXSTS
0245 09          609        IN      A,P1           ;READ PORT 1
0246 02          610        OUT     DBB,A          ;PUT DATA IN DBBOUT
0247 440C        611        JMP     MNLOOP         ;LOOP
                 612 ;
                 613        END

USER SYMBOLS
ASCMSK 007F   B110   013E   B1200  014A   B300   0142   B600   0146   BFFLG  0004   CMD    0100   CMD1   0112
CMD2   0121   CMDJ1  0212   DATA   0153   DATRDY 0008   ERROR  013A   ERRST  0131   FESTS  0040   FRAMER 0010
INIT   0200   INT1   0008   IODATA 0167   IODIR  0040   IOER   0127   IOFLAG 023C   IOFLG  0080   MARK   0004
MKOUT  0001   MNLOOP 020C   MNLP1  0214   MOUT   0019   OVRUN  0020   OVSTS  0080   RCV    001B   RCV1   0026
RCV2   0033   RCV3   0038   RCV4   0042   RCV5   004D   RCV6   0051   RCV7   0055   RCV8   0060   RCV9   0064
REQFLG 0002   RESET  0000   RETURN 0000   RSTERR 003F   RXF    022E   RXFLG  0001   RXINTL 0000   RXO    0235
RXRDY  021E   RXRDY1 0223   RXRDY2 0226   RXSTS  0003   SBIT   0008   SPACE  00FB   SPOUT  00FE   SRTBIT 0003
SRTFLG 0002   STPEND 0024   STTIMR 014C   TICOUT 0040   TICSRT 0004   TIMCON 0020   TIMINT 0007   TXBIT  0020
TXEND  0028   TXFLG  0001   TXINT  0008   TXOUT  0012   TXSTS  0005   TXTIC  0003   XMIT   0068   XMT1   0081
XMT2   008C   XMT3   0097   XMT4   009C   XMT5   00A8   ZERO   0000

ASSEMBLY COMPLETE,  NO ERRORS
```

```
LOC  OBJ        SEQ        SOURCE STATEMENT

                 1 ;
                 2 ;TEST ROUTINE WHICH OUTPUTS THE ASCII CHARACTER SET TO THE
                 3 ;UPI TRANSMITTER AND DISPLAYS ON THE 80/30 CONSOLE ANY
                 4 ;CHARACTERS RECEIVED BY THE UPI RECEIVER.
                 5 ;
                 6 ;INPUTS: NOTHING
                 7 ;OUTPUTS: CHARACTERS TO CONSOLE
                 8 ;CALLS: NOTHING
                 9 ;
4000            10        ORG     4000H
00DF            11 MODE53 EQU     0DFH            ;8253 CONTROL PORT
00DC            12 CNT0   EQU     0DCH            ;8253 CNT 0 PORT
00E5            13 CMD    EQU     0E5H            ;UPI COMMAND PORT
00E5            14 STATUS EQU     0E5H            ;UPI STATUS PORT
00E4            15 DBBIN  EQU     0E4H            ;UPI DBBIN PORT
00E4            16 DBBOUT EQU     0E4H            ;UPI DBBOUT PORT
0020            17 TXINT  EQU     20H             ;TXINT MASK
0001            18 OBF    EQU     01H             ;OBF MASK
0002            19 IBF    EQU     02H             ;IBF MASK
00ED            20 STAT51 EQU     0EDH            ;8251 STATUS PORT
00EC            21 DATA51 EQU     0ECH            ;8251 DATA PORT
0001            22 TXRDY  EQU     01H             ;8251 TXRDY MASK
                23 ;
4000 3E36       24 START: MVI     A,36H           ;8253 CNT0 MODE WORD
4002 D3DF       25        OUT     MODE53          ;8253 CONTROL PORT
4004 3E10       26        MVI     A,10H           ;DIVIDE BY 16D
4006 D3DC       27        OUT     CNT0            ;8253 CNT0 PORT LSB
4008 3E00       28        MVI     A,00H           ;
400A D3DC       29        OUT     CNT0            ;8253 CNT0 PORT MSB
400C 0620       30        MVI     B,20H           ;INITIALIZE OUTPUT CHR
400E 3E10       31        MVI     A,10H           ;CONFIGURE COMMAND - 1200 BAUD
4010 D3E5       32        OUT     CMD             ;UPI COMMAND PORT
4012 DBE5       33 POLL1: IN      STATUS          ;READ UPI STATUS
4014 E621       34        ANI     TXINT OR OBF    ;TEST TXINT AND OBF
4016 CA1240     35        JZ      POLL1           ;WAIT UNTIL ONE IS SET
4019 DBE5       36        IN      STATUS          ;READ UPI STATUS AGAIN
401B E601       37        ANI     OBF             ;WAS IT OBF?
401D C23840     38        JNZ     RX              ;YES, GO DO RECEIVER
                39                                ;NO, MUST BE TRANSMITTER
4020 78         40        MOV     A,B             ;GET NEXT CHR FOR OUTPUT
4021 D3E4       41        OUT     DBBIN           ;OUTPUT TO UPI DBBIN
4023 FE5A       42        CPI     'Z'             ;WAS IT LAST CHR?
4025 CA3340     43        JZ      NEWB            ;YES, RESET REG. B
4028 04         44        INR     B               ;OTHERWISE, INC B
4029 DBE5       45 POLL2: IN      STATUS          ;TEST IF IBF STILL SET
402B E602       46        ANI     IBF             ;TEST IBF
402D C22940     47        JNZ     POLL2           ;WAIT UNTIL IBF=0
4030 C31240     48        JMP     POLL1           ;BEFORE LOOKING AT STATUS AGAIN
                49 ;
4033 0620       50 NEWB:  MVI     B,20H           ;RESET REG. B
4035 C32940     51        JMP     POLL2           ;GO BACK
                52 ;
```

```
LOC  OBJ        SEQ        SOURCE STATEMENT

4038 DBE4        53 RX:     IN      DBBOUT        ;READ DBBOUT FOR RECEIVED CHR
403A 4F          54         MOV     C,A           ;SAVE IT IN C
403B DBED        55 RX1:    IN      STAT51        ;READ 8251 STATUS
403D E601        56         ANI     TXRDY         ;TEST TXRDY
403F CA3840      57         JZ      RX1           ;WAIT UNTIL READY
4042 79          58         MOV     A,C           ;GET CHR
4043 D3EC        59         OUT     DATA51        ;OUTPUT CHR TO CONSOLE
4045 C31240      60         JMP     POLL1         ;GO TEST UPI AGAIN
                 61 ;
                 62 END

PUBLIC SYMBOLS


EXTERNAL SYMBOLS


USER SYMBOLS
CMD    A 00E5   CNT0   A 000C   DATA51 A 00EC   DBBIN  A 00E4   DBBOUT A 00E4   16F    A 0002   MODE53 A 000F
NEWB   A 4033   OBF    A 0001   POLL1  A 4012   POLL2  A 4029   RX     A 4038   RX1    A 403B   START  A 4000
STAT51 A 00ED   STATUS A 00E5   TXINT  A 0020   TXRDY  A 0001

ASSEMBLY COMPLETE,  NO ERRORS
```

PROGRAMMABLE KEYBOARD INTERFACE

- **Simultaneous Keyboard and Display Operations**
- **Interface Signals for Contact and Capacitive Coupled Keyboards**
- **128-Key Scanning Logic**
- **10.7msec Matrix Scan Time for 128 Keys and 6MHz Clock**
- **Eight Character Keyboard FIFO**
- **N-Key Rollover with Programmable Error Mode on Multiple New Closures**
- **Sixteen or Eight Character Seven-Segment Display Interface**
- **Right or Left Entry Display RAM**
- **Depress/Release Mode Programmable**
- **Interrupt Output on Key Entry**

This application is a general purpose programmable keyboard and display interface device designed for use with 8-bit microprocessors like the MCS-80 and MCS-85. The keyboard portion can provide a scanned interface to 128-key contact or capacitive-coupled keyboards. The keys are fully debounced with N-key rollover and programmable error generation on multiple new key closures. Keyboard entries are stored in an 8-character FIFO with overrun status indication when more than 8 characters are entered. Key entries set an interrupt request output to the master CPU.

The display portion of the UPI-41A provides a scanned display interface for LED, incandescent and other popular display technologies. Both numeric displays and simple indicators may be used. The UPI-41A has a 16×4 display RAM which can be

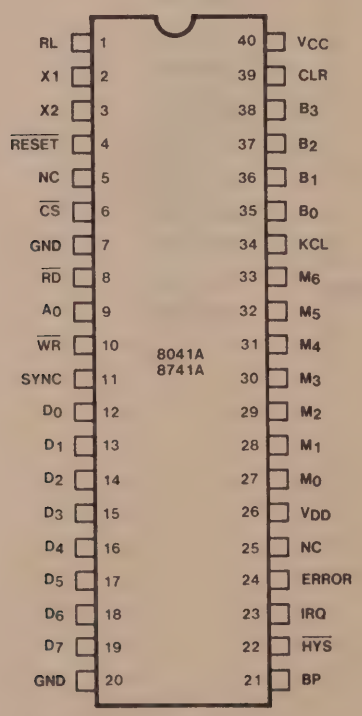


Figure 1. Pin Configuration

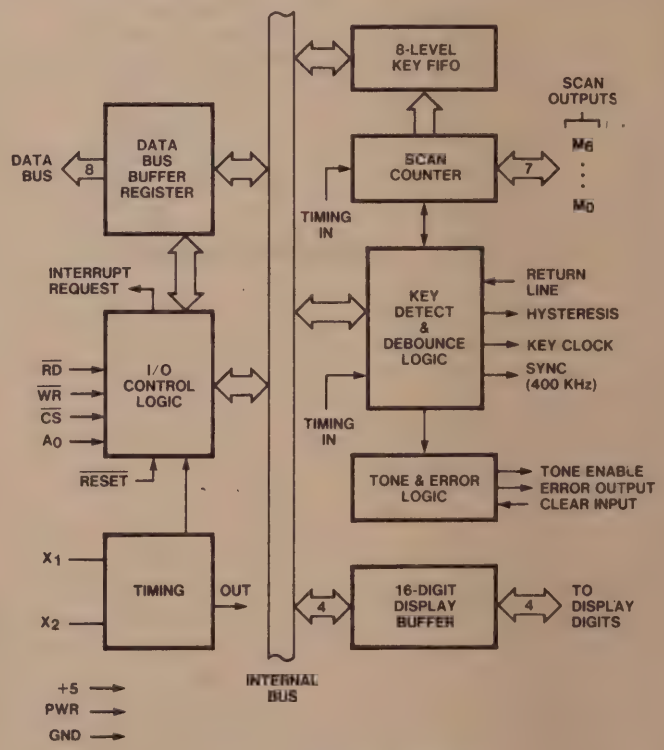


Figure 2. Block Diagram

loaded or interrogated by the CPU. Both right entry calculator and left entry typewriter display formats are possible. Both read and write of the display RAM can be done with auto increment of the display RAM address.

ORDERING INFORMATION:

This part may be ordered as an 8041A with ROM code number 8278. The source code is available through Insite.

Throughout this application of the UPI-41A, it will be referred to by its ROM code number, 8278. The 8278 is packaged in a 40-pin DIP. The following is a brief functional description of each pin.

PRINCIPLES OF OPERATION

The following is a description of the major elements of the Programmable Keyboard/Display interface device. Refer to the block diagram in Figure 1.

I/O Control and Data Buffers

The I/O control section uses the $\overline{CS}$, A_0, $\overline{RD}$, and $\overline{WR}$ lines to control data flow to and from the various internal registers and buffers (see Table 2). All data flow to and from the 8278 is enabled by $\overline{CS}$. The 8-bits of information being transferred by the CPU is identified by A_0. A logic one means information is command or status. A logic zero means the information is data. $\overline{RD}$ and $\overline{WR}$ determine the direction of data flow through the Data Bus Buffer (DBB). The

Table 1. Pin Description

Signal	Pin. No.	Type	Name and Function
D_0-D_7	12-19	I/O	**Data Bus:** Three-state, bi-directional data bus lines used to transfer data and commands between the CPU and the 8278.
$\overline{WR}$	10	I	**Write:** Write strobe which enables the master CPU to write data and commands between the CPU and the 8278.
$\overline{RD}$	8	I	**Read:** Read strobe which enables the master CPU to read data and status from the 8278 internal registers.
$\overline{CS}$	6	I	**Chip Select:** Chip select input used to enable reading and writing to the 8278.
A_0	9	I	**Control/Data:** Address input used by the CPU to indicate control or data.
$\overline{RESET}$	4	I	**Reset:** A low signal on this pin resets the 8278.
X_1, X_2	2,3	I	**Freq. Reference Inputs:** Inputs for crystal, L-C or external timing signal to determine internal oscillator frequency.
IRQ	23	O	**Interrupt Request:** Interrupt Request Output to the master CPU. In the keyboard mode the IRQ line goes low with each FIFO read and returns high if there is still information in the FIFO or an ERROR has occurred.
M_0–M_6	27-33	O	**Matrix Scan Lines:** Matrix scan outputs. These outputs control a decoder which scans the key matrix columns and the 16 display digits. Also, the Matrix scan outputs are used to multiplex the return lines from the key matrix.
RL	1	I	**Keyboard Return Line:** Input from the multiplexer which indicates whether the key currently being scanned is closed.
$\overline{HYS}$	22	O	**Hysteresis:** Hysteresis output to the analog detector. (Capacitive keyboard configuration). A "0" means the key currently being scanned has already been recorded.
KCL	34	O	**Key Clock:** Key Clock output to the analog detector (capacitive keyboard configuration) used to reset the detector before scanning a key.
SYNC	11	O	**Output Clock:** High frequency (400 kHz) output signal used in the key scan to detect a closed key (capacitive keyboard configuration).
B_0-B_3	35-38	O	**Display Outputs:** These four lines contain binary coded decimal display information synchronized to the keyboard column scan. The outputs are for multiplexed digital displays.
ERROR	24	O	**Error Signal:** This line is high whenever two new key closures are detected during a single scan or when too many characters are entered into the keyboard FIFO. It is reset by a system RESET pulse or by a "1" input on the CLR pin or by the CLEAR ERROR command.
CLR	39	I	**Clear Error:** Input used to clear an ERROR condition in the 8278.
BP	21	O	**Tone Enable:** Tone enable output. This line is high for 10ms following a valid key closure; it is set high and remains high during an ERROR condition.
V_{CC}, V_{DD}	40,26	I	**Power:** +5 volt power input: +5V ± 10%.
GND	20,7	I	**Ground:** Signal ground.

DBB register is a bi-directional 8-bit buffer register which connects the internal 8278 bus buffer register to the external bus. When the chip is not selected ($\overline{CS}$ = 1) the DBB is in the high impedance state. The DBB acts as an input when ($\overline{RD}$, $\overline{WR}$, $\overline{CS}$) = (1, 0, 0) and an output when ($\overline{RD}$, $\overline{WR}$, $\overline{CS}$) = (0, 1, 0).

Table 2. I/O Control and Data Buffers

$\overline{CS}$	A_0	$\overline{WR}$	$\overline{RD}$	Condition
0	0	1	0	Read DBB Data
0	1	1	0	Read STATUS
0	0	0	1	Write Data to DBB
0	1	0	1	Write Command to DBB
1	X	X	X	Disable 8278 Bus, High Impedance

Scan Counter

The scan counter provides the timing to scan the keyboard and display. The four MSB's (M_3-M_6) scan the display digits and provide column scan to the keyboard via a 4 to 16 decoder. The three LSB's (M_0-M_2) are used to multiplex the row return lines into the 8278.

Keyboard Debounce and Control

The 8278 system configuration is shown in Figure 3. The rows of the matrix are scanned and the outputs are multiplexed by the 8278. When a key closure is detected, the debounce logic waits about 12 msec to check if the key remains closed. If it does, the address of the key in the matrix is transferred into a FIFO buffer.

FIFO and FIFO Status

The 8278 contains an 8×8 FIFO character buffer. Each new entry is written into a successive FIFO location and each is then read out in the order of entry. A FIFO status register keeps track of the number of characters in the FIFO and whether it is full or empty. Too many reads or key entries will be recognized as an error. The status can be read by a $\overline{RD}$ with $\overline{CS}$ low and A_0 high. The status logic also provides a IRQ signal to the master processor whenever the FIFO is not empty.

Display Address Registers and Display RAM

The Display Address registers hold the address of the word currently being written or read by the CPU and the two 4-bit nibbles being displayed. The read/write addresses are programmed by CPU command. They also can be set to auto increment after each read or write. The display RAM can be directly read by the CPU after the correct mode and address is set. Data entry to the display can be set to either left or right entry.

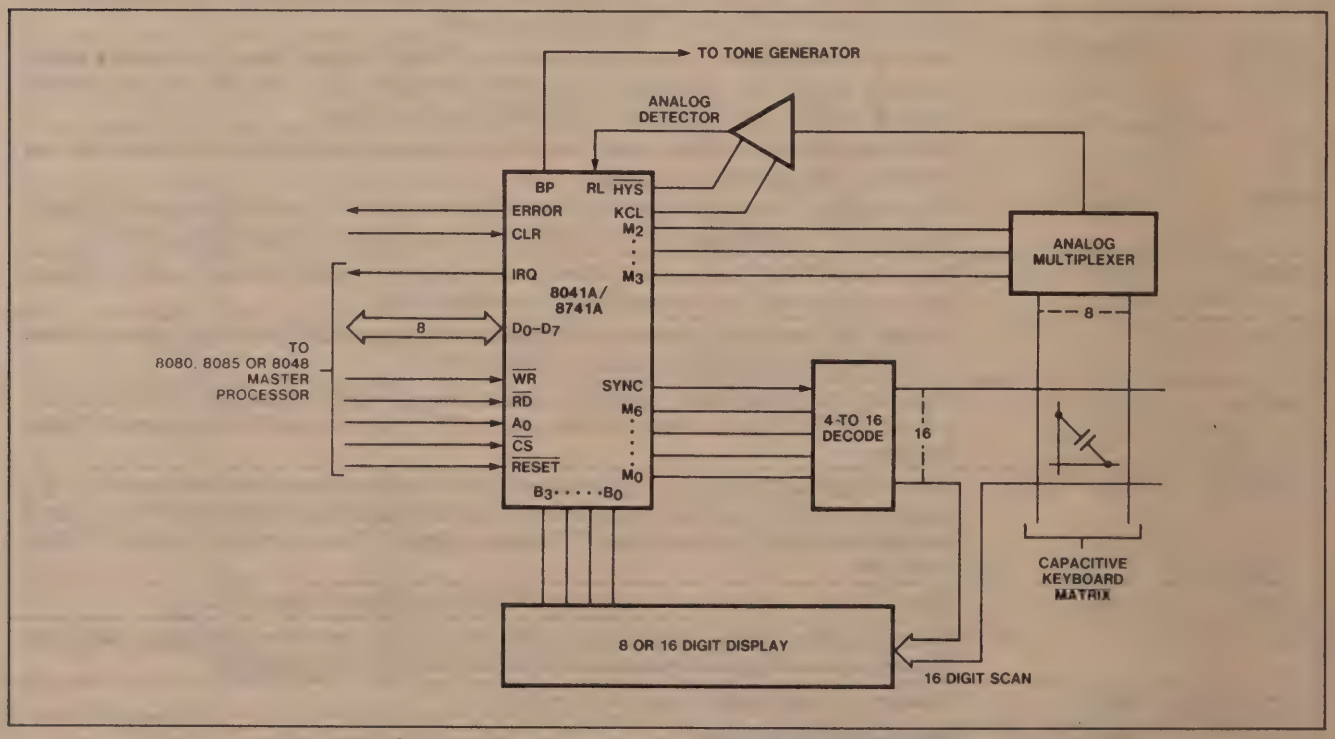


Figure 3. System Configuration for Capacitive-Coupled Keyboard

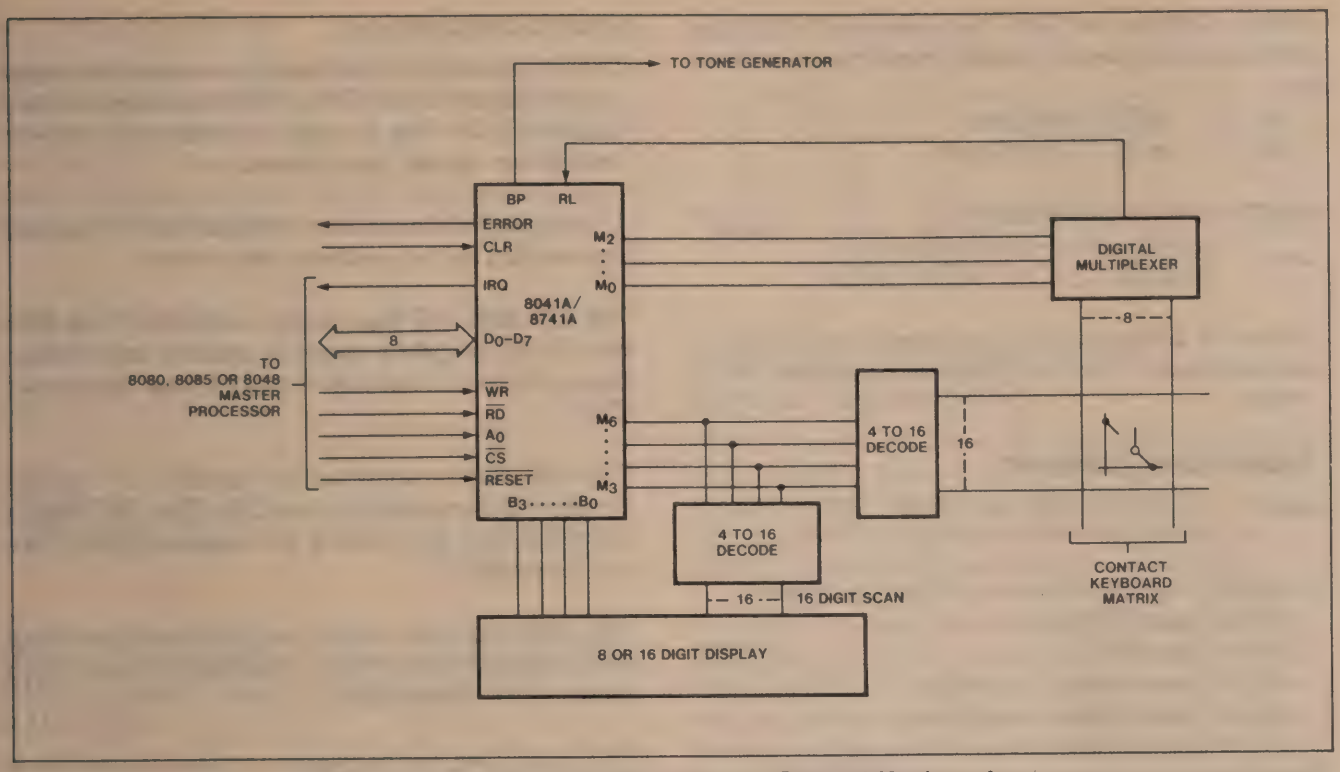
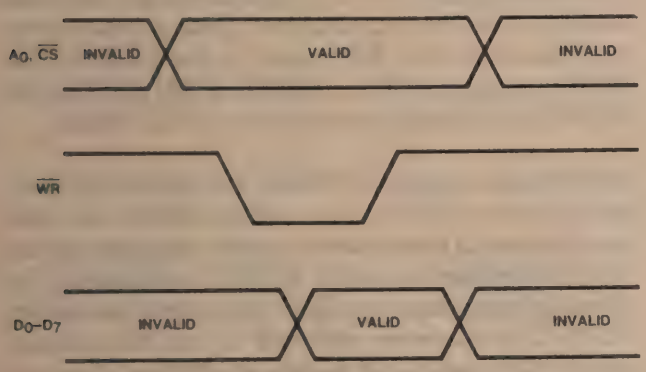


Figure 4. System Configuration for Contact Keyboard

COMMANDS

The 8278 operating mode is programmed by the master CPU using the A_0, WR and D_0-D_7 inputs as shown below:



The master CPU presents the proper command on the D_0-D_7 data lines with A_0 =1 and then sends a $\overline{WR}$ pulse. The command is latched by the 8278 on the rising edge of the $\overline{WR}$ and is decoded internally to set the proper operating mode. See the 8041A/8741A data sheet for timing details.

Command Summary

KEYBOARD/DISPLAY MODE SET

CODE	0	0	0	N	E	I	D	K

Where the mode set bits are defined as follows:

K—the keyboard mode select bit
- 0—normal key entry mode
- 1—special function mode: Entry on key closure and on key release

D—the display entry mode select bit
- 0—left display entry
- 1—right display entry

I—the interrupt request (IRQ) output enable bit.
- 0—enable IRQ output
- 1—disable IRQ output

E—the error mode select bit
- 0—error on multiple key depression
- 1—no error on multiple key depression

N—the number of display digits select
- 0—16 display digits
- 1—8 display digits

NOTE:
The default mode following a RESET input is all bits zero:

0	0	0	0	0	0	0	0

READ FIFO COMMAND

CODE	0	1	0	0	0	0	0	0

READ DISPLAY COMMAND

CODE	0	1	1	AI	A_3	A_2	A_1	A_0

Where AI indicates Auto Increment and A_3-A_0 is the address of the next display character to be read out.

AI = 1 AUTO increment
AI = 0 no AUTO increment

WRITE DISPLAY COMMAND

CODE	1	0	0	AI	A_3	A_2	A_1	A_0

Where AI indicates Auto Increment and A_3-A_0 is the address of the next display character to be written.

CLEAR/BLANK COMMAND

CODE	1	0	1	UD	BD	CD	CF	CE

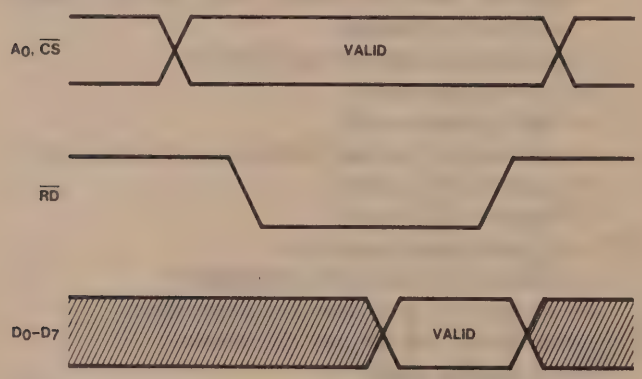
Where the command bits are defined as follows:

CE = Clear ERROR
CF = Clear FIFO
CD = Clear Display to all High
BD = Blank Display to all High
UD = Unblank Display

The display is cleared and blanked following a Reset.

Status Read

The status register in the 8278 can be read by the master CPU using the A_0, $\overline{RD}$, and D_0-D_7 inputs as shown below:



The 8278 places 8-bits of status information on the D_0-D_7 lines following (A_0, $\overline{CS}$, $\overline{RD}$) = 1, 0, 0 inputs from the master.

Status Format

S_3	S_2	S_1	S_0	B	KE	IBF	OBF
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

Where the status bits are defined as follows:

IBF = Input Buffer Full Flag
OBF = Output Buffer Full Flag
KE = Keyboard Error Flag (multiple depression)
B = BUSY Flag
S_3-S_0 = FIFO Status

STATUS DESCRIPTION

The S_3-S_0 status bits indicate the number of entries (0 to 8) in the 8-level FIFO. A FIFO overrun will lock status at 1111. The overrun condition will prevent further key entries until cleared.

A multiple key closure error will set the KE flag and prevent further key entries until cleared.

The IBF and OBF flags signify the status of the 8278 data buffer registers used to transfer information (data, status or commands) to and from the master CPU.

The IBF flag is set when the master CPU writes Data or Commands to the 8278. The IBF flag is cleared by the 8278 during its response to the Data or Command.

The OBF flag is set when the 8278 has output data ready for the master CPU. This flag is cleared by a master CPU Data READ.

The Busy flag in the status register is used as a LOCKOUT signal to the master processor during response to any command or data write from the master.

The master must test the Busy flag before each read (during a sequence) to be sure that the 8278 is ready with valid DATA.

The ERROR and TONE outputs from the 8278 are set high for either type of error. Both types of error are cleared by the CLR input, by the CLEAR ERROR command, or by a reset. The FIFO and Display buffers are cleared independently of the Errors.

FIFO status is used to indicate the number of characters in the FIFO and to indiate whether an error has occurred. Overrun occurs when the entry of another character into a full FIFO is attempted. Underrun occurs when the CPU tries to read an empty FIFO. The character read will be the last one entered. FIFO status will remain at 0000 and the error condition will not be set.

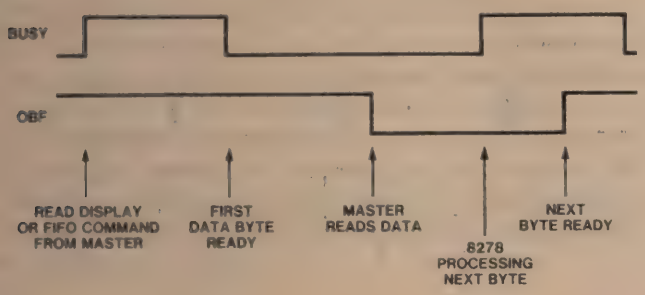
Data Read

The master CPU can read DATA from the 8278 FIFO or Display buffers by using the A_0, $\overline{RD}$, and D_0-D_7 inputs.

The master sends a $\overline{RD}$ pulse with A_0 = 0 and CS = 0 and the 8278 responds by outputting data on lines D_0-D_7. The data is strobed by the trailing edge of $\overline{RD}$.

DATA READ SEQUENCE

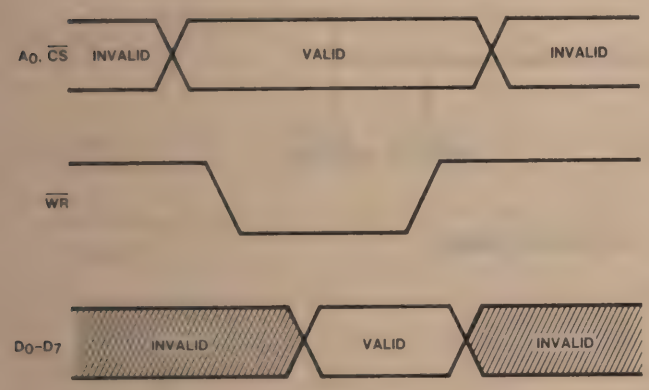
Before reading data, the master CPU must send a command to select FIFO or Display data. Following the command, the master must read STATUS and test the BUSY flag and the OBF flag to verify that the 8278 has responded to the previous command. A typical DATA READ sequence is as follows:



After the first read following a Read Display or Read FIFO command, successive reads may occur as soon as OBF rises.

Data Write

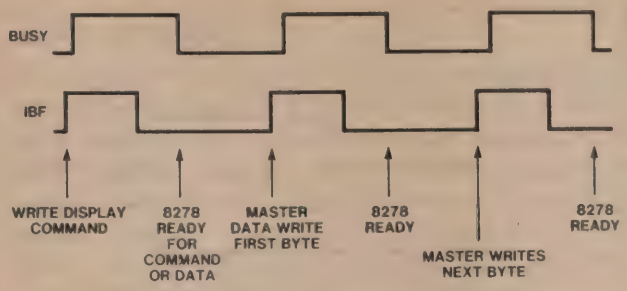
The master CPU can write DATA to the 8278 Display buffers by using the A_0, $\overline{WR}$ and D_0-D_7 inputs as follows:



The master CPU presents the Data on the D_0-D_7 lines with A_0=0 and then sends a $\overline{WR}$ pulse. The data is latched by the 8278 on the rising edge of $\overline{WR}$.

DATA WRITE SEQUENCE

Before writing data to the 8278, the master CPU must first send a command to select the desired display entry mode and to specify the address of the next data byte. Following the commands, the master must read STATUS and test the BUSY flag (B) and IBF flag to verify that the 8278 has responded. A typical sequence is shown below.



INTERFACE CONSIDERATIONS

Scanned Keyboard Mode

With N-key rollover each key depression is treated independently from all others. When a key is depressed the debounce logic waits for a full scan of 128 keys and then checks to see if the key is still down. If it is, the key is entered into the FIFO.

If two key closures occur during the same scan the ERROR output is set, the KE flag is set in the Status word, the TONE output is activated and IRQ is set, and no further inputs are accepted. This condition is cleared by a high signal on the CLEAR input or by a system $\overline{RESET}$ input or by the CLEAR ERROR command.

In the special function mode both the key closure and the key release cause an entry to the FIFO. The release is entered with the MSB=1.

Any key entry triggers the TONE output for 10ms.

The $\overline{HYS}$ and KCL outputs enable the analog multiplexer and detector to be synchronized for interface to capacitive coupled keyboards.

Data Format

In the scanned keyboard mode, the code entered into the FIFO corresponds to the position or address of the switch in the keyboard. The MSB is relevant only for special function keys in which code "0" signifies closure and "1" signifies release. The next four bits are the column count which indicates which column the key was found in. The last three bits are from the row counter.

BIT	7	6	5	4	3	2	1	0
	SF	M_6	M_5	M_4	M_3	M_2	M_1	M_0

1 FOR SPECIAL FUNCTION MODE AND KEY RELEASED
0 FOR KEY DEPRESSED

Display

Display data is entered into a 16×4 display register and may be entered from the left, from the right or

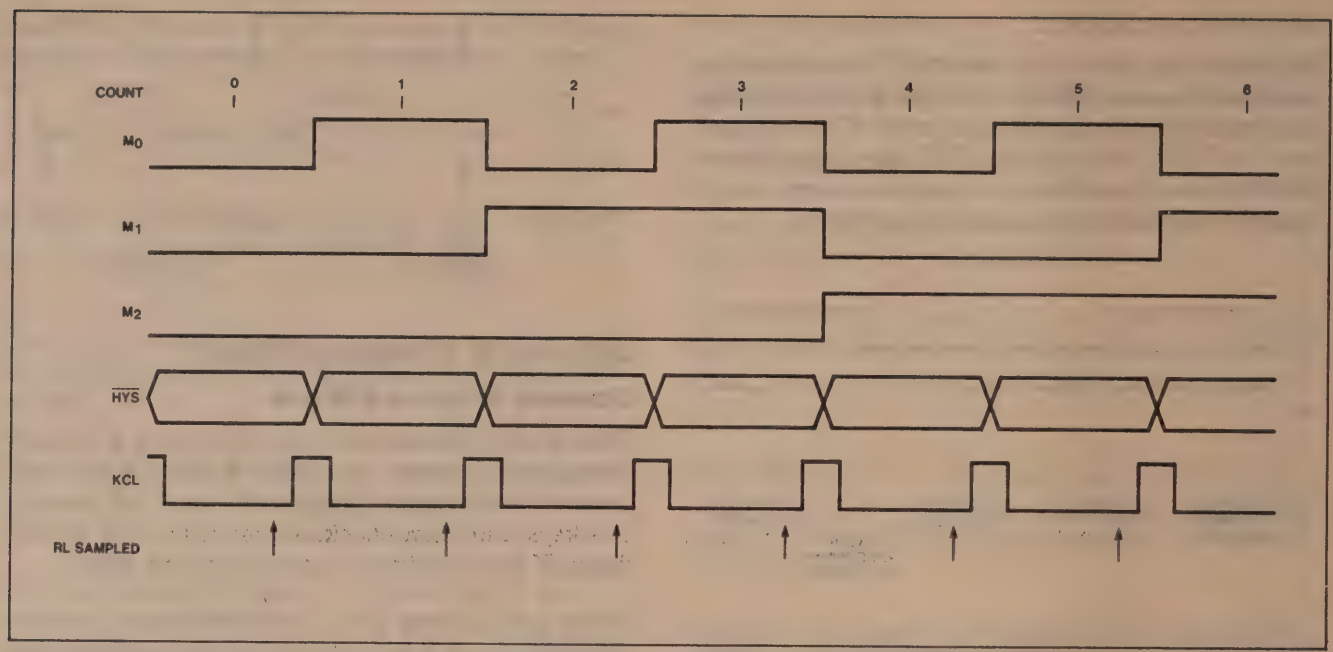


Figure 5. Keyboard Timing

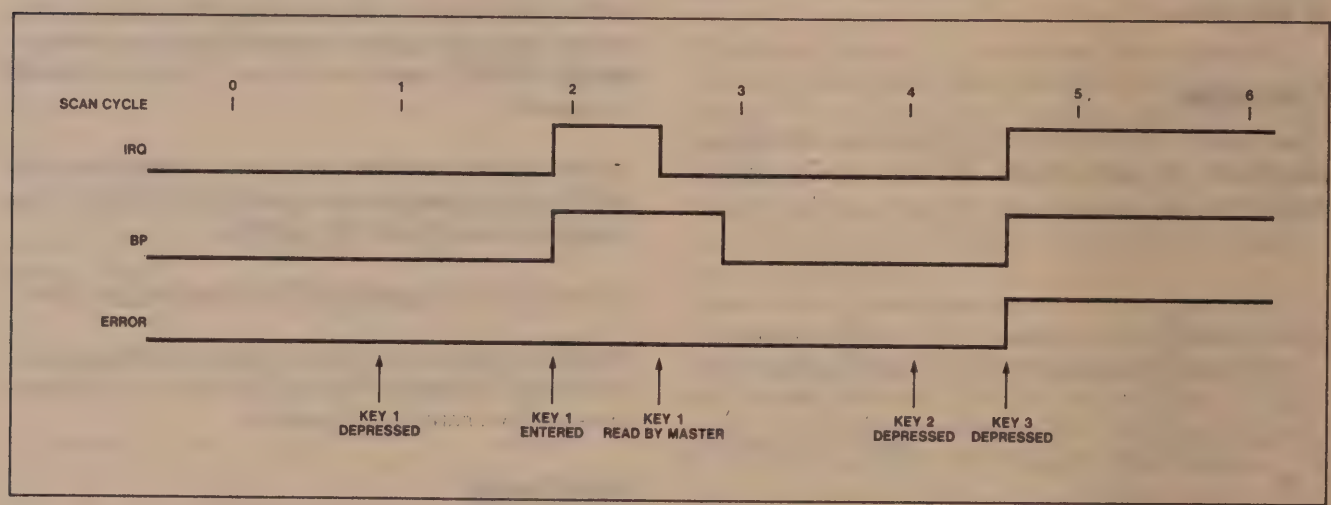


Figure 6. Key Entry and Error Timing

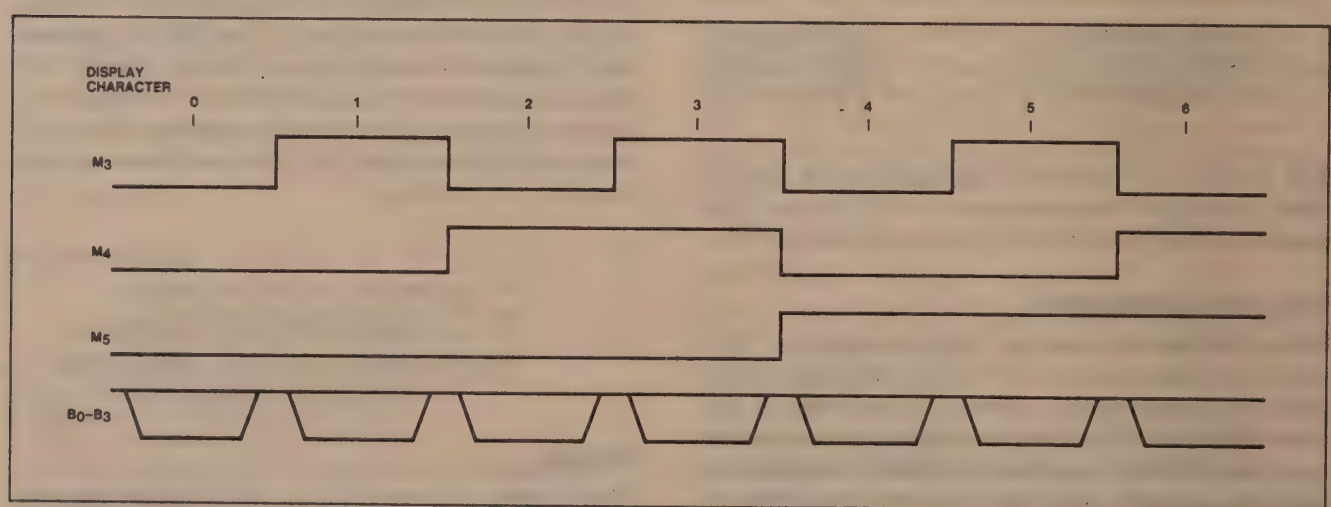


Figure 7. Display Timing

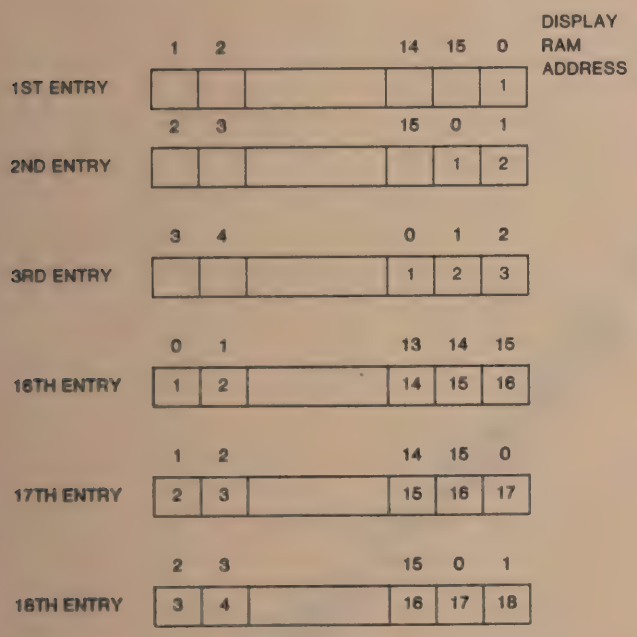
into specific locations in the display register. A new data character is put out on B_0-B_3 each time the M_6-M_3 lines change (i.e., once every 0.75ms with a 6 MHz crystal). Data is blanked during the time the column select lines change by raising the display outputs. Output data is positive true.

LEFT ENTRY

The left entry mode is the simplest display format in that each display position in the display corresponds to a byte (or nibble) in the Display RAM. ADDRESS 0 in the RAM is the left-most display character and ADDRESS 15 is the right-most display character. Entering characters from position zero causes the display to fill from the left. The 17th character is entered back in the left-most position and filling again proceeds from there.

RIGHT ENTRY

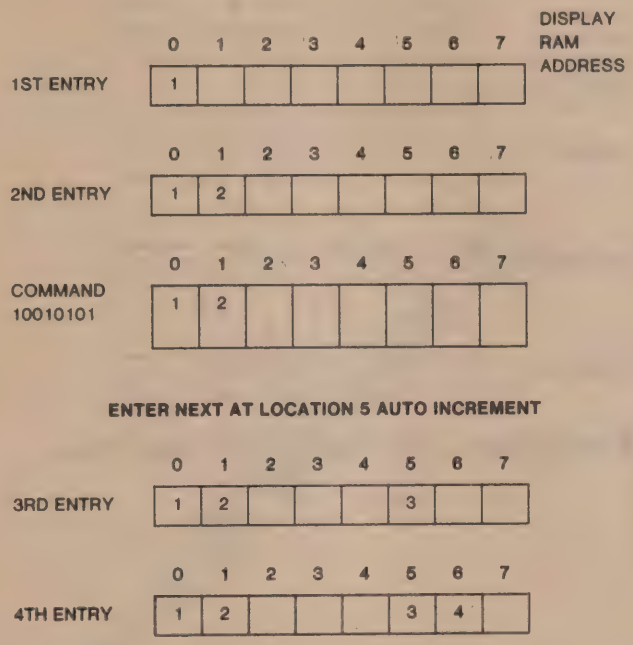
Right entry is the method used by most electronic calculators. The first entry is placed in the right-most display character. The next entry is also placed in the right-most character after the display is shifted left one character. The left-most character is shifted off the end and is lost.



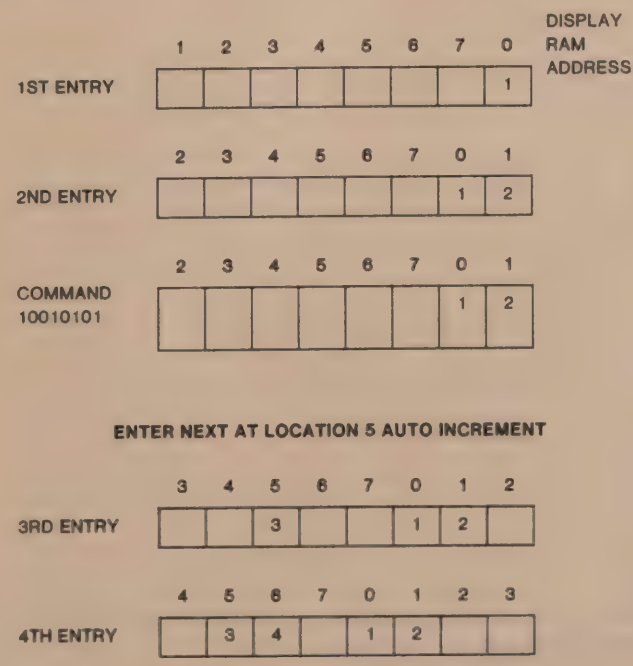
Note that now the display position and register address do not correspond. Consequently, entering a character to an arbitrary position in the Auto Increment mode may have unexpected results. Entry starting at Display RAM ADDRESS 0 with sequential entry is recommended. A Clear Display command should be given before display data is entered if the number of data characters is not equal to 16 (or 8) in this mode.

AUTO INCREMENT

In the Left Entry mode, Auto Incrementing causes the address where the CPU will next write to be incremented by one and the character appears in the next location. With non-Auto Incrementing the entry is both to the same RAM address and display position. Entry to an arbitrary address in the Left Entry—Auto Increment mode has no undesirable side effects and the result is predictable:



In the Right Entry mode, Auto Incrementing and non-Incrementing have the same effect as in the Left Entry except that the address sequence is interrupted.



Starting at an arbitrary location operates as shown below.

	0	1	2	3	4	5	6	7	DISPLAY RAM ADDRESS
COMMAND 10010101									

ENTER NEXT AT LOCATION 5 AUTO INCREMENT

	1	2	3	4	5	6	7	0
1ST ENTRY					1			

	2	3	4	5	6	7	0	1
2ND ENTRY				1	2			

8TH ENTRY	4	5	6	7	8	1	2	3

9TH ENTRY	5	6	7	8	9	2	3	4

Entry appears to be from the initial entry point.

intel®

APPLICATION NOTE

AP-161

September 1983

Complex Peripheral Control with the UPI-42

© INTEL CORPORATION, 1983

NOVEMBER 1983
ORDER NUMBER: 230795-001

COMPLEX PERIPHERAL CONTROL WITH THE UPI-42

TABLE OF CONTENTS

INTRODUCTION
DOT MATRIX PRINTING
THE PRINTER MECHANISM
HARDWARE INTERFACE
TECHNICAL BACKGROUND
SOFTWARE
Introduction
Functional Overview
Memory and Register Allocation
Description of Functional Blocks and Flowcharts
CONCLUSION

APPENDICES
Appendix A. Software Listing
Appendix B. Printer Enhancements
Appendix C. Printer Mechanism Drive Circuit Schematics

FIGURES

1. UPI-42 Pin Configuration
2. UPI-42 Block Diagram
3. UPI-41A, 42 Functional Block Diagram
4. Character E in 5 x 7 Dot Matrix Format
5. Carriage Stepper Motor Assembly
6. Print Head Solenoid Assembly
7. Hardware Interface Block Diagram
8. Hardware Interface Schematic
9. UPI-42 and 8243 I/O Port Map
10. Stepper Motor Step Sequence Waveforms
11. Carriage Stepper Motor Step Sequence
12. Paper Feed Stepper Motor Step Sequence
13. Carriage Stepper Motor Drive Timing
14. Carriage Stepper Motor Predetermined Time Constants
15. Paper Feed Stepper Motor Predetermined Time Constants
16. PTS Lags PT Timing
17. PTS Leads PT Timing
18. Components of Print Head Assembly Line Motion and Printing
19. Data Memory Allocation Map
20. Register Bank 0 Register Assignment
21. Register Bank 0 Status Byte Flag Assignments

22. Register Bank 1
Register Assignment
23. Register Bank 1 Status
Byte Flag Assignments
24. Program Memory Allocation Map
25. ASCII Character Code TEST
Output and Print Example
26. Carriage Stepper Motor
Phase/Step Data

FLOW CHARTS

1. Main Program Body
2. Power-On/Reset Initialization
3. Home Print Head Assembly
4. External Status Switch Check
5. Character Buffer Fill
6. Carriage Stepper Motor Drive
and Line Printing
7. Carriage Stepper Motor
Acceleration Time Storage
8. Process Characters for Printing
9. Translate Character-to-Dots
10. Decelerate Carriage
Stepper Motor
11. Paper Feed Stepper Motor Drive

Addtional sources of information on Intel's UPI devices;

"UPI User's Manual"
Includes the following Application Notes;
Programmable Keyboard Interface
Using the 8295 Dot Matrix Printer Controller
An 8741A/8041A Digital Cassette Controller

"8048 Family Applications Handbook"

"1983 Microprocessor and Peripheral Handbook"

"MCS-48 and UPI-41A/42 Assembly Language Manual"

"Specifications for Impact Dot Matrix Printer Model-3210", Epson, Jan 8, 1981

INTRODUCTION

The UPI-42 is the newest member of Intel's Universal Peripheral Interface (UPI) microcomputer family. It represents a significant growth in UPI capabilities resulting in a broader spectrum of applications. The UPI-42 incorporates twice the EPROM/ROM of the UPI-41A, 2048 vs 1024 bytes, twice the RAM, 128 vs 64 bytes, and operates at a maximum speed twice that of the UPI-41A, i.e. 12 MHz vs 6 MHz. The ROM based 8042 and the EPROM based 8742 provide more highly integrated solutions for complex stepping motor and dot matrix printer applications. Those applications previously requiring a microprocessor plus a UPI chip can now be implemented entirely with the UPI-42.

The software features of the UPI-42, such as indirect Data and Program Memory addressing, two independent and selectable 8 byte register banks, and directly software testable I/O pins, greatly simplify the external interface and software flow. The software and hardware design of the UPI-42 allows a complex peripheral to be controlled with a minimum of external hardware.

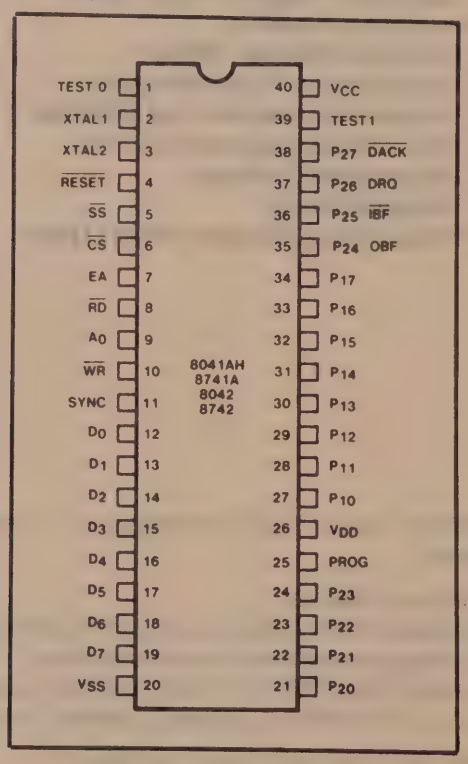


Figure 1. UPI-42 Pin Configuration

Many microcomputer systems need real time control of peripheral devices such as a printer, keyboard, complex motor control or process control. These medium speed but still time consuming tasks require a fair amount of system software overhead. This processing burden can be reduced by using a dedicated peripheral control processor

Until recently, the dedicated control processor approach was usually not cost effective due to the large number of components needed; CPU, RAM, ROM, I/O, and Timer/Counters. To help make the approach more cost effective, in 1977 Intel introduced the UPI-41 family of Universal Peripheral Interface controllers consisting of an 8041 (ROM) device and an 8741 (EPROM) device. These devices integrated the common microprocessor system functions into one 40 pin package. The UPI-42 family, consisting of the 8042 and 8742, further extends the UPI's cost effectiveness through more memory and higher speed.

Another member of the UPI family is the Intel 8243 Input/Output Expander chip. This chip provides the UPI-41A and UPI-42 with up to 16 additional independently programmable I/O lines, and interfaces directly to the UPI-41A/42. Up to seven 8243s can be cascaded to provide over 100 I/O lines.

The UPI is a single chip microcomputer with a standard microprocessor interface. The UPI's architecture, illustrated in Figure 3, features on-chip program memory, ROM (8041A/8042) or EPROM (8741A/8742), data memory (RAM), CPU, timer/counter, and I/O. Special interface registers are provided which enable the UPI to function as a peripheral to an 8-bit central processor.

Using one of the UPI devices, the designer simply codes his proprietary peripheral control algorithm into the UPI device itself, rather than into the main system software. The UPI device then performs the peripheral control task while the host processor simply issues commands and transfers data. With the proliferation of microcomputer systems, the use of UPIs or slave microprocessors to off load the main system microprocessor has become quite common.

This Application Note describes how the UPI-42 can be used to control dot matrix printing and the printer mechanism, using stepper motors for carriage/print head assembly and paper feed motion. Previous Intel Application Notes AP-27, AP-54, and AP-91 describe using intelligent processors and peripherals to control single solenoid driven printer mechanisms with 80 character line buffering and bidirectional printing. This Application Note expands on these previous themes and extends the concept of complex device control by incorporating full 80 character line buffering, bidirectional printing, as well as drive and feedback control of two four phase stepper motors.

The Application Note assumes that the reader is familiar with the 8042/8742 and 8243 Data Sheets, and UPI-41A/42 Assembly Language. Although some background information is included, it also assumes a basic understanding of stepper motors and dot matrix printer mechanisms. A complete software listing is included in Appendix A.

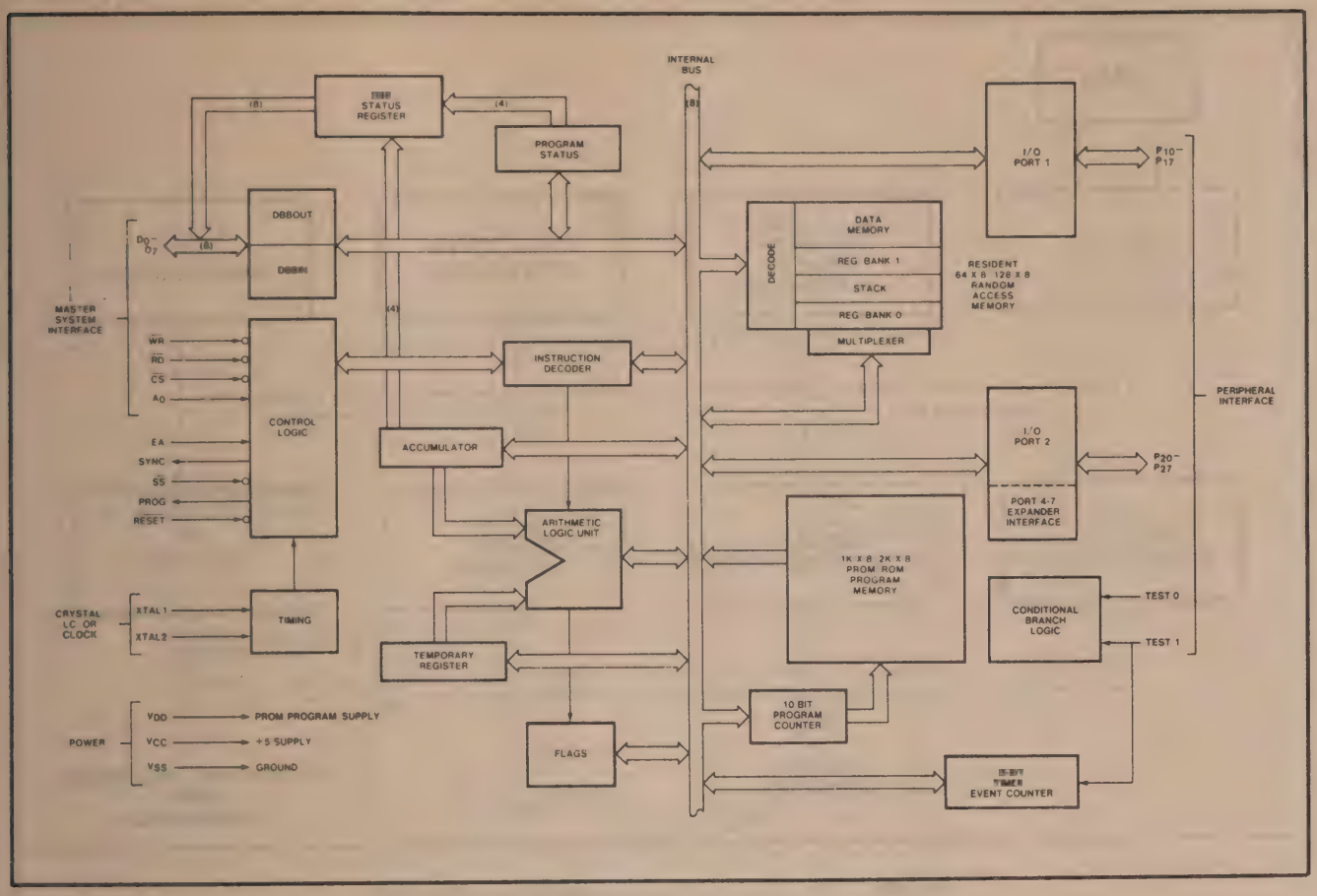


Figure 2. UPI-42 Block Diagram

DOT MATRIX PRINTING

A dot matrix printer print head typically consists of seven to nine solenoids, each of which drives a stiff wire, or hammer, to impact the paper through an inked ribbon. Characters are formed by firing the solenoids to form a matrix of "dots" (impacts of the wires). Figure 4 shows how the character "E" is formed using a 5 x 7 matrix. The columns are labeled C1 through C5, and the rows R1 through R7. The print head moves left-to-right across the paper, so that at time T1 the head is over column C1. The character is formed by activating the proper solenoids as the print head sweeps across the character position.

Dot matrix printers are a cost effective way of providing good quality hard copy output for microcomputer systems. There is an ever increasing demand for the moderately priced printer to provide more functionality with improved cost and performance. Using stepper motors to control the paper feed and carriage/print head assembly motion is one way of enabling the dot matrix printer to provide more capabilities, such as expanded or contracted characters, dot or line graphics, variable line and character spacing, and subscript or superscript printing.

However, stepper motors require fairly complex contol algorithms. Previous solutions involved the use of a main CPU, UPI, RAM, ROM, and I/O onboard the peripheral. The CPU acted as supervisor and used parallel processing to achieve accurate stepper motor control via a UPI, character buffering via the I/O device, RAM, and ROM. The CPU performed real-time decoding of each character into a dot matrix pattern. This Application Note demonstrates that the increased memory and performance of the UPI-42 facilitates integrating these control functions to reduce the cost and component count.

THE PRINTER MECHANISM

The printer mechanism used in this application is the Epson Model 3210. It consists of four basic subassemblies; the chassis or frame, the paper feed mechanism and stepper motor, the carriage motion mechanism and stepper motor, and the print head assembly.

The paper feed mechanism is a tractor feed type. It accomodates up to 8.5 inch wide paper (not including tractor feed portion). There is no platen as such; the paper is moved through the paper guide by two sprocketed wheels mounted on a center sprocket shaft. The sprocket shaft is driven by a four phase stepper motor. The rotation of the stepper motor is transmitted to the sprocket shaft through a series of four reduction gears.

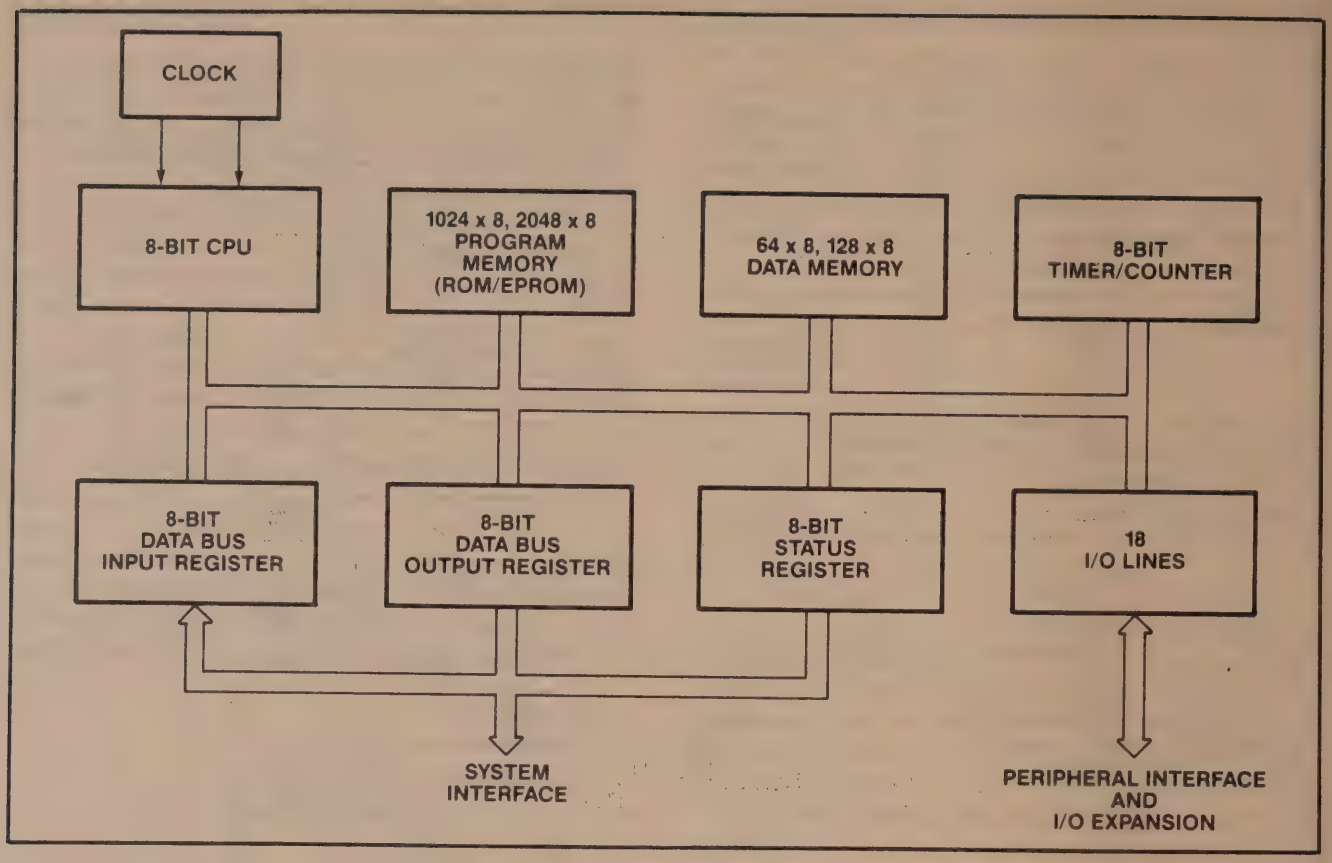


Figure 3. UPI-41A, 42 Functional Block Diagram

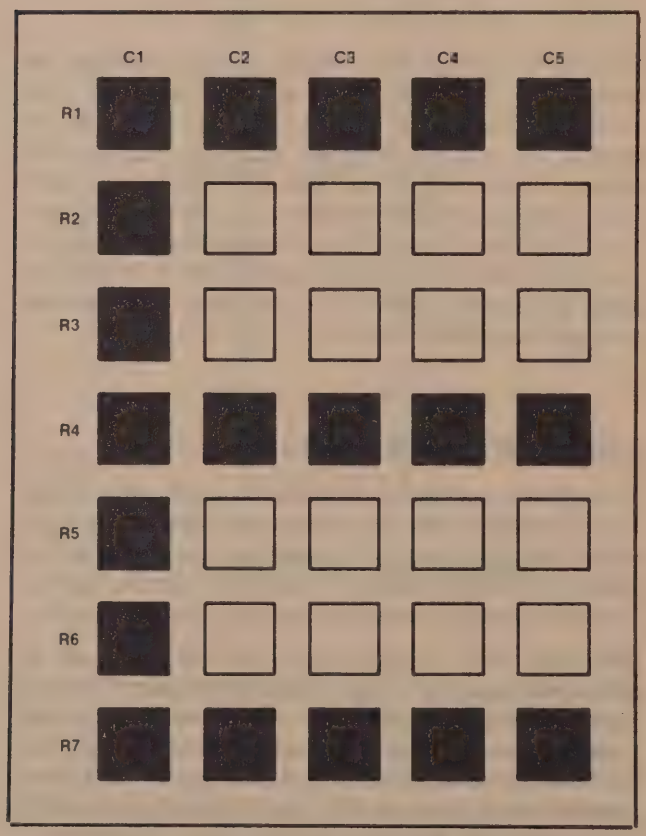


Figure 4. Character E in 5 x 7 Dot Matrix Format

The carriage motion mechanism consists of another four phase stepper motor which controls the left-to-right or right-to-left print head assembly motion. The print speed is 80 CPS maximum. Both the speed of the stepper motor and the movement of the print head assembly are independently controllable in either direction. The rotation of the stepper motor is converted to the linear motion of the print head assembly via a series of reduction gears and a toothed drive belt. The drive belt also controls a second set of reduction gears which advances the print ribbon as the print head assembly moves.

Two optical sensors provide feedback information on the carriage assembly position and speed. The first of these optical sensors, called the 'HOME RESET' or HR, is mounted near the left-most physical position to which the print head assembly can move. As the print head assembly approaches the left-most position, a flange on the print head assembly interferes with the light source and sensor, causing the output of the sensor to shift from a logic level one to zero. The right-most printer position is monitored in software rather than by another optical sensor. The right-most print position is a function of the number of characters printed and the distance required to print them.

The second optical sensor, called the 'PRINT TIMING SIGNAL' or PTS, provides feedback on carriage stepper motor velocity and relative position within a

STEPPER MOTOR
STEPPER MOTOR FRAME
OPTICAL SENSOR
ENCODER DISK
PRINT HEAD ASSEMBLY TOOTHED DRIVE BELT
REDUCTION GEARS

Figure 5. Carriage Stepper Motor Assembly

given step of the motor. The feedback is generated by the optical sensor as an "encoder disk" moves across it. Figure 5 illustrates the carriage stepper motor, optical sensor, encoder disk and reduction gears, and drive belt assembly. The optical sensor outputs a pulse train with the same period as the phase shift signal used to drive the stepper, but slightly out of phase with it when the motor is at a constant speed (see Software Functional Block: Phase Shift Data for additional details). The disk acts as a timing wheel, providing feedback to the UPI software of the carriage speed, position, and optimum position for energizing the print head solenoids. The two optical sensors are monitored under software and provide the critical feedback needed to control the print head assembly and paper feed motion accurately. The process of stepper motor drive and control via feedback signals is called "closed loop" stepper motor control, and is covered in more detail in the software discussion.

The print head assembly consists of nine solenoids and nine wires or hammers. Figure 6 illustrates a print head assembly. The available dot matrix measures 9 x 9. This large matrix enables the Epson 3210 print mechanism to print a variety of character fonts, such as expanded or contracted characters, as well as line or block graphics (see Appendix B, Printer Enhancements). It also facilitates printing lower case ASCII characters with "lower case descenders." That is to say, certain lower case letters (e.g. y, p, etc.) will print below the bottom part of all upper case letters.

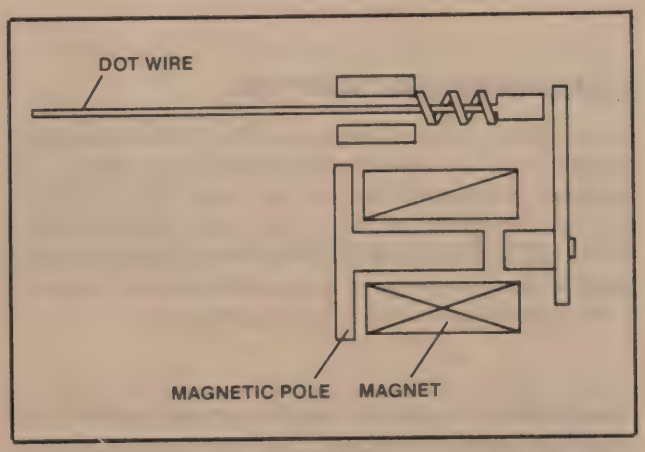


Figure 6. Print Head Solenoid Assembly

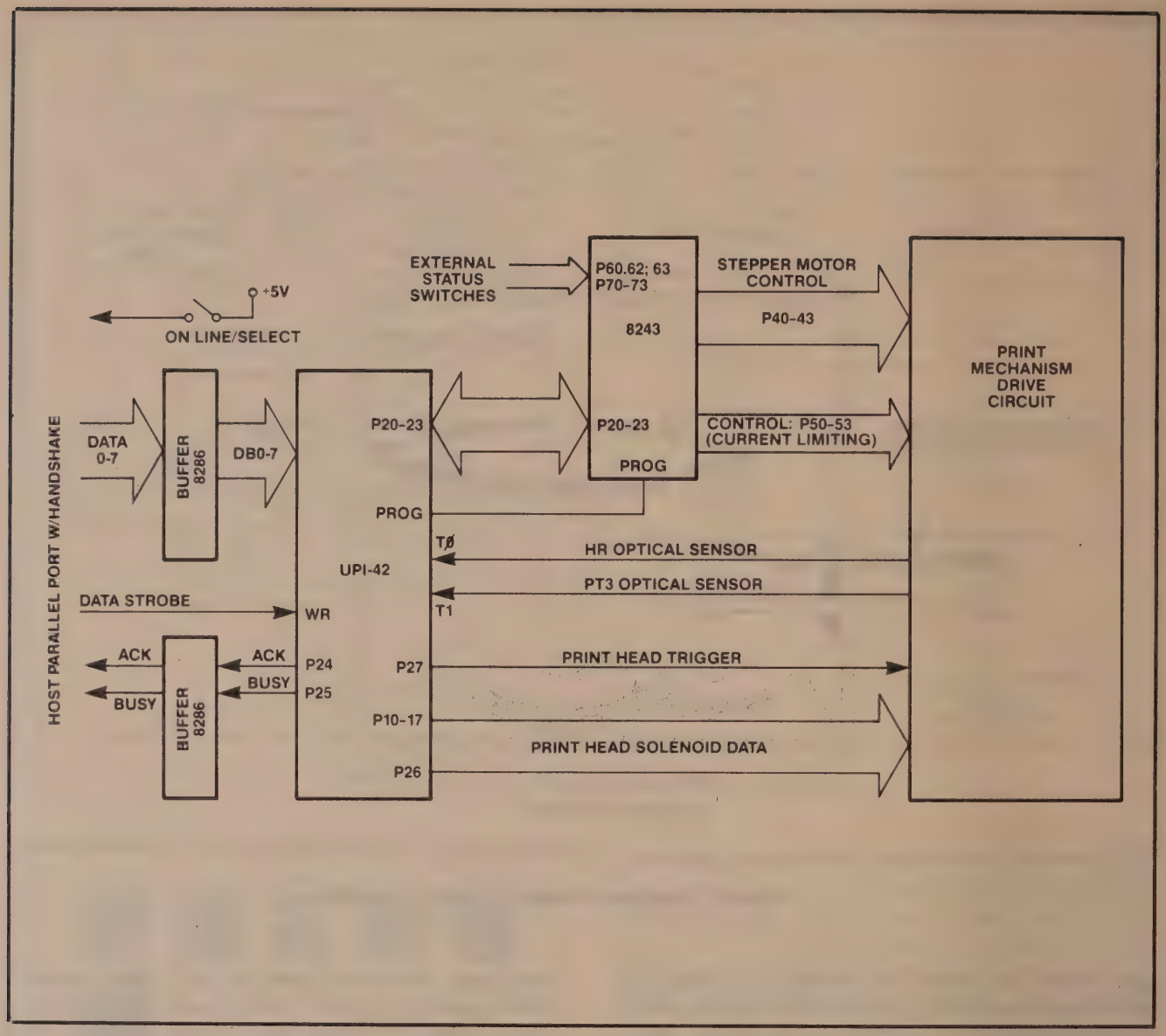


Figure 7. Hardware Interface Block Diagram

HARDWARE DESCRIPTION

Figure 7 shows a block diagram of the UPI-42 and 8243 interface to the printer mechanism drive circuit. A complete schematic is shown in Figure 8. The UPI-42 provides all signals necessary to control character buffering and handshaking, paperfeed and carriage motion stepper motor timing, print head solenoid activation, and monitoring of external status switches.

The Epson 3210 printer mechanism manual recommends a specific interface circuit to provide proper drive levels to the stepper motors windings and print head solenoids. The hardware interface used for this Application Note followed those recomendations exactly (see Appendix C, Printer Mechanism Drive Circuit Schematics).

I/O Ports

The lower half of the UPI-42 Port 2, pins 0-3, provides an interface to the 8243 I/O expander. The PROG pin of the UPI-42 is used as a strobe to clock address and data information via the Port 2 interface. The extra 16 I/O lines of the 8243 become PORTS 4, 5, 6, and 7 to the UPI software. Combined, the UPI-42 and 8243 provide a total of 28 independently programmable I/O line. These lines are used as follows:

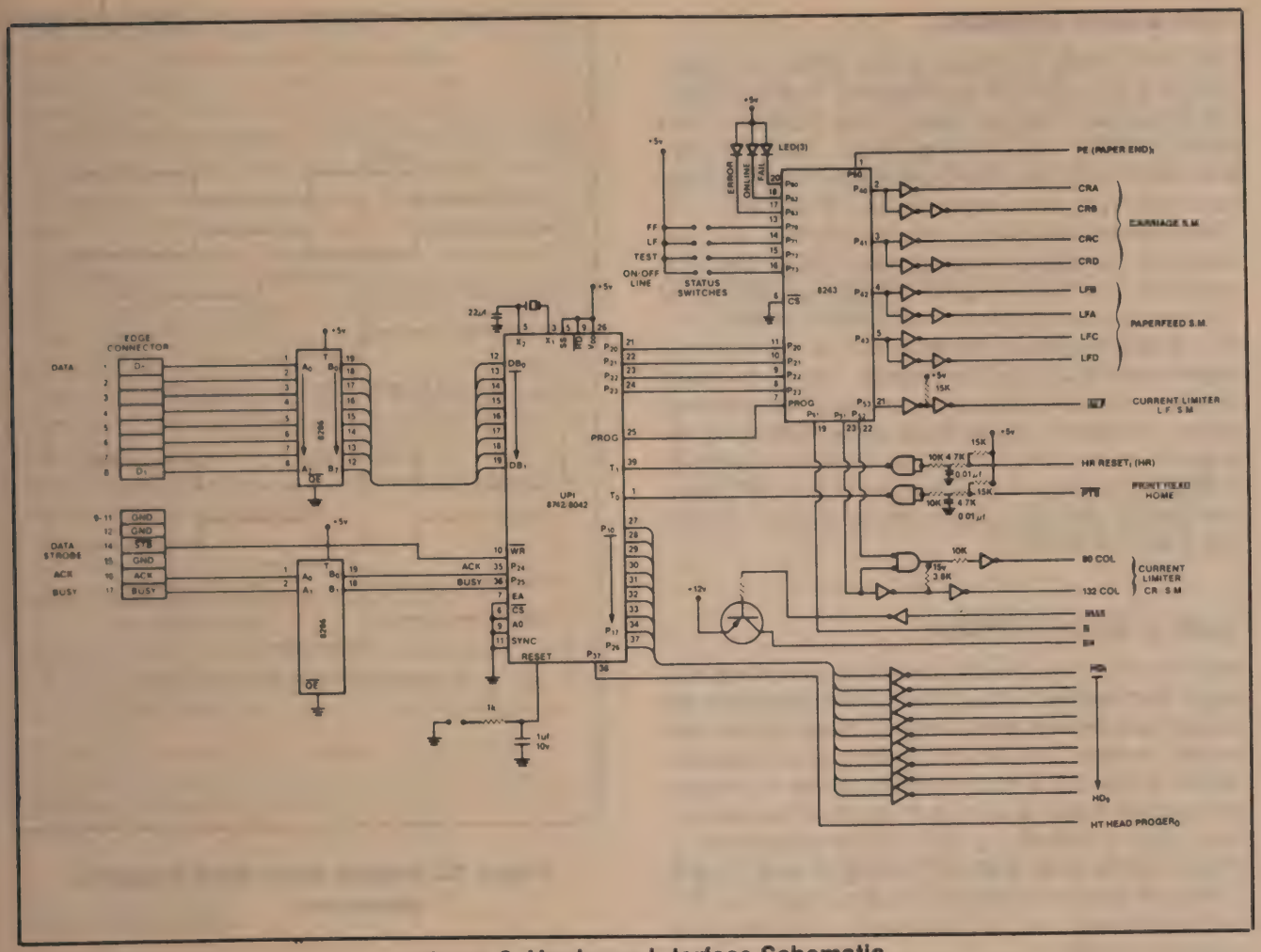


Figure 8. Hardware Interface Schematic

Port	No of lines	Bits	I/O	Description
1	8	0-7	O	Character dot column data to print head solenoids
2	1	6	O	(same)
2	1	7	O	Print head solenoid trigger
2	2	4,5	O	Host system data transfer handshaking (ACK/BUSY)
4	4	0-3	O	Carriage & paper feed stepper motors
5	3	1-3	O	Stepper motor select and current limiting
5	1	0	I	Paper End sense
6	1	1	O	Print head trigger reset
6	3	0,2,3	-	(unused)
7	5	0-3	I	External status switches; (LF, FF, TEST, ON/OFF Line)

Figure 9. UPI-42 and 8243 I/O Port Map

Note: The notation used in the balance of this Application Note, when referring to a port number and a particular pin or bit, is Port 23 rather than Port 2 bit 3.

The two printer mechanism optical sensors, discussed in the Printer Mechanism discription, are tied to the two "Test Input" pins, T0 and T1, of the UPI-42 through a buffer circuit for noise supression. These inputs are directly testable in software.

Host System Interface

The host system interfaces to the printer through a parallel port to the UPI-42 Data Bus. Four handshaking signals are used to control data transfer; Data Strobe (STB/), Acknowledge (ACK), Busy (BUSY), and Online or Select. The Data Strobe line of the host parallel port is tied directly to the UPI-42 WR/ pin. This provides a low going pulse on the UPI-42 WR/ pin whenever a data byte is written to the UPI-42. The ACK and BUSY handshake signals are tied to two UPI-42 I/O port lines for software control of data transfer. The "On Line" handshake signal is tied to a single-pole single-throw fixed position switch, which externally enables or disables character transfer from the host system. Characters transmitted to the UPI-42 by the host are loaded into the UPI-42 Data Bus Buffer In (DBBIN) register, and the Input Buffer Full (IBF) interrupt and UPI-42 status flag are set (see Figure 9. UPI-42 and 8243 I/O Ports).

Stepper Motor Interface

Port 4 (41-43) of the 8243, provides both carriage and paper feed stepper motor phase shift signals to the printer mechanism drive circuit. Each of the two stepper motors is driven by 2 two phase excitation signals (4 phases). Figure 10 shows the wave form for each stepper motor. Each signal consists of two components (Sig. 1 A/B & Sig. 2 C/D) 180 degrees out of phase with the other. Each of these signal pairs (A/B & C/D) is 90 degrees out of phase with the other pair. For each signal pair, one port line supplies both halves by using an inverter.

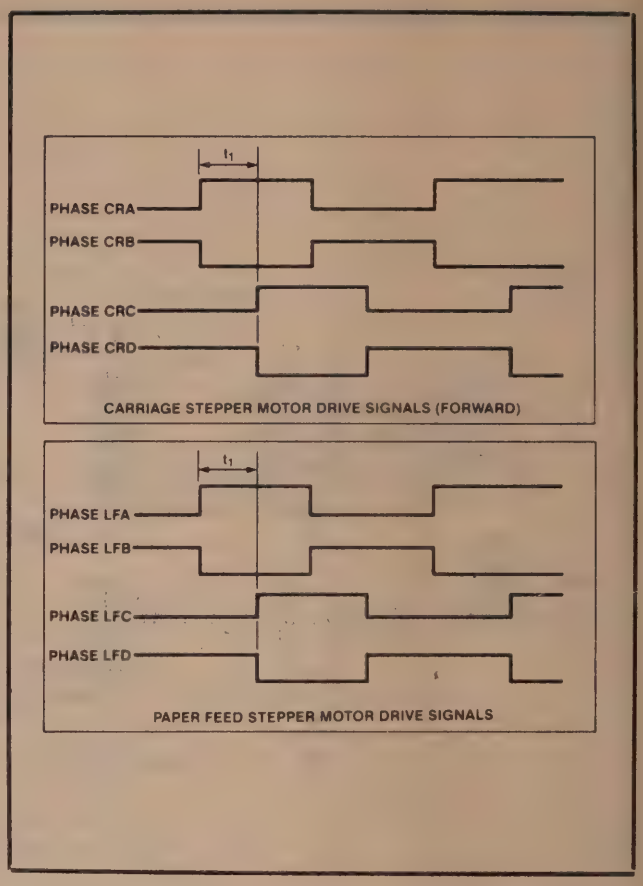


Figure 10. Stepper Motor Step Sequence Waveforms

Each of the resulting eight stepper motor drive signals is interfaced to a discrete drive transistor through an inverter. The emitter of the drive transistor is tied to the open collector of the inverter to provide high current sinking capability for the drive transistor. Each half of the motor winding is tied to the collector of the drive transistor (see Appendix C, Printer Mechanism Drive Circuit Schematic).

Each stepper motor requires two current levels for operation. These levels are called "Rush" current and "Hold" current. Rush current refers to the high current required to cause the rotor to rotate within its windings as the polarity of the power applied to the windings is changing. Each change in the polarity of the power applied to the motor windings is called a step or phase shift. Hold current refers to the low level of current required to stabilize and maintain the rotor in a fixed position when the the polarity applied to the windings is not changing. Hold current is simply Rush current with a current limiting transistor switched in. Switching from Hold to Rush current "selects" or enables that stepper motor to move with the next step signal output. In the balance of this Application Note, the term "select" will be used to refer to turning on Rush current, and "deselect" will refer to switching to Hold currrent.

Three 8243 port lines are dedicated to the select/deselect control of the two stepper motors. One line is for the paper feed stepper motor, and two lines are for the carriage motion stepper motor (80 and 132 column). These lines are labeled SLF, 80Col, and 132Col, and are 8243 PORT 53, 52, and 51, respectively.

By varying the voltage applied to the stepper motor biasing circuit and the current, it is possible to vary the distance the motor moves the print head assembly with each step. Enabling one of two different voltage biasing levels, and changing the timing rate at which the motor is stepped, facilitates either 80 or 132 character column printing. Only 80 character column printing is implemented in the software design. Appendix B, Printer Enhancements, details the software algorithm for handling 132 character printing.

Print Head Interface

A total of eleven I/O lines are used to control the print head solenoids and solenoid firing (see Figure 9 above). Nine are used for character dot data, one for the Print Head Trigger, and one for Reset of the Print Head Trigger circuit. Each of the nine character dot data lines is buffered by an open collector hex inverter.

The Print Head Trigger output is tied to the Trigger input of a 555 Monostable Multivibrator. The output pulse generated by the 555 triggers the print head solenoids to fire. The 555 Output pulse width is independent of the input trigger waveform. The pulse width is determined by an RC network across the 555 inputs and the voltage level applied to the Control Voltage 555 input. The 555 Output is tied to the base of a PNP transistor through an inverter, biased in a normally off configuration. The PNP transistor supplies enough drive to pull up the open collector inverter on each print head solenoid line, Port 10-17 and 26. The 555 output pulse momentarily enables the print head solenoid line open collector inverter output, turning on the solenoid drive transistor, and firing the print head hammer. The 555 Ouput pulse width is approximately 400 us. Further details of the print head firing operation can be found in the software description below.

Miscellaneous Interface Signals

The 8243 Port 5 pin 0 is tied to the Paper End Detector, a reed switch located on the printer paper guide. This sensor detects when the paper is nearly exhausted.

Three LED status lights complete the hardware interface design. One status light is used for each of the following: Power ON/OFF, On/Off Line, and Out of Paper.

BACKGROUND

Before a detailed discussion of the software begins, a few terms and software functions referenced throughout the software need introduction.

A. What is a Stepper Motor?

A stepper motor has the ability to rotate in either direction as well as start and stop at predetermined angular positions. The stepper motor's shaft (rotor) moves in precise angular increments for each input step. The displacement is repeated for each input step command, accurately positioning the rotor for a given number and sequence of steps.

The stepper motor controls position, velocity, and direction. The accuracy of stepper motors is generally 5 percent of one step. The number of steps in each revolution of the shaft varies, depending on the intended application.

B. Open/Closed Loop Stepper Motor Drive and Control

The carriage stepper motor is closed loop driven. The paper feed stepper motor is open loop driven.

There are two major types of stepper motor control known by the broad headings of open and closed loop. Open loop is simply continuous pulses to drive the motor at a predetermined rate based on the voltage, current, and the timing of the step pulses applied. Closed loop control is characterized by continuous monitoring of the stepper motor, through feedback signals, and adjusting the motor's operation based upon the feedback received.

C. Stepper Motor Drive Phase Shift or Step Sequence

Each change in the polarity of the power applied to the motor windings is called a step or phase shift. The sequence of the steps or phase shifts, and the pattern of polarity changes output to the stepper motor, determines the direction of rotation.

Figure 10 shows the waveforms for each of the two stepper motors. Figure 11 lists the step sequence for carriage motor clockwise rotation, which moves the print head assembly Left-to-Right. Figure 11 also lists the step sequence for counterclockwise rotations; the print head assembly moves Right-to-Left. Figure 12 lists the step sequence for the paper feed stepper motor clockwise drive. The phase sequence, for either stepper motor, may begin at any point within the sequence list, but must then continue in order.

Step No.	A-Step	B-Step	C-Step	D-Step
1	On	Off	Off	On
2	On	Off	On	Off
3	Off	On	On	Off
4	Off	On	Off	On

Carriage stepper motor rotates clockwise
Print head assembly moves from left to right

Step No.	A-Step	B-Step	C-Step	D-Step
1	On	Off	On	Off
2	On	Off	Off	On
3	Off	On	Off	On
4	Off	On	On	Off

Carriage stepper motor rotates counter clockwise
Print head assembly moves from right to left

Figure 11. Carriage Stepper Motor Step Sequence

Step No.	A-Step	B-Step	C-Step	D-Step
1	On	Off	On	Off
2	On	Off	Off	On
3	Off	On	Off	On
4	Off	On	On	Off

Figure 12. Paper Feed Stepper Motor Step Sequence

C. Acceleration and Deceleration of Stepper Motors

The carriage stepper motor starts from a fixed position, accelerates to a constant speed, maintains constant speed, and then decelerates to a fixed position. Printing may occur from the time and position the print head assembly reaches constant speed, until the time and position the print head assembly begins to decelerate from constant speed. Whether printing occurs during any carriage stepper motor drive sequence is controlled by software. Figure 18, below, illustrates these components of print head assembly line motion.

Due to inertia, a finite time interval and angular displacement is required to accelerate a stepper motor to its full speed. Conversely, deceleration must begin some time before the final angular position. The time interval and angular displacement of the carriage stepper motor translates into the distance the print head assembly travels before it reaches a constant speed. The distance traveled during acceleration is constant. The distance the print head assembly travels during deceleration must be the same as the distance traveled during acceleration in order to accurately align the character dot columns from one line to the next.

E. Stepper Motor Predetermined Time Constant

Whenever the stepper motor is stepped, or energized, the angular velocity of the rotor is greater than the constant speed which is ultimately required. This is called "overshoot." The frictional load of the carriage assembly (motor rotor, reduction gears, drive belt and print head assembly, or paper feed sprocket shaft and wheels) provides damping or frictional load. Damping slows the motor to less than the required constant speed and is called "undershoot" (see Figure 13, Carriage Stepper Motor Drive Timing). A constant rate of speed is achieved through the averaging of the overshoot and undershoot within each step.

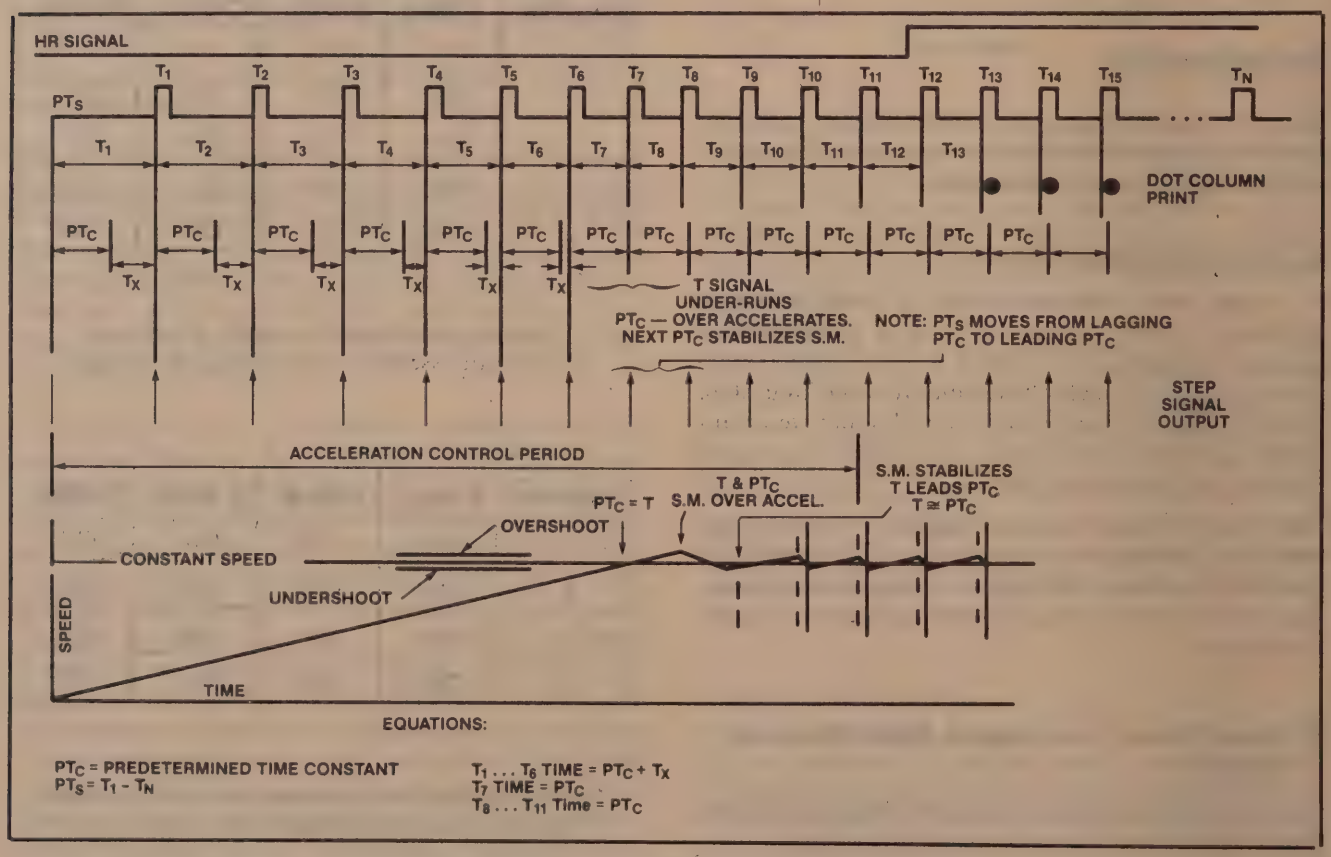


Figure 13. Carriage Stepper Motor Drive Timing

The Predetermined Time (PT) Constant is the time required to average the overshoot and undershoot of the particular stepper motor for a desired constant rate of speed. The PT also is the time required to move the print head assembly a specific distance, acounting for both overshoot and undershoot of the stepper motor.

Changing the Predetermined Time Constant changes the angular displacement of the stepper motor rotor, this in turn changes the output. Figure 14 lists the Time Constants for both standard and condensed character printing. Figure 15 lists the paper feed stepper motor Time Constants used for various line spacing formats. This Application Note implements standard character print and paper feed (6 lines per inch) Time Constants. See Appendix B, Printer Enhancements, for details on implementing non-standard Time Constants.

Character mode	Predetermined time	
Standard or Enlarged Character	2.08ms	+10% -4%
Condensed Character	4.16ms	+10% -4%

Figure 14. Carriage Stepper Motor Predetermined Time Constants

Paper feed pitch

0.12mm(1/216") /1 pulse
4.23mm(1/6") /36 pulses
3.18mm(1/8") /27 pulses
2.82mm(1/9") /24 pulses

Paper feed time

150ms/4.23mm	Approx. 6.6 lines/s (continuous feed)
113ms/3.18mm	Approx. 8.8 lines/s (continuous feed)
100ms/2.82mm	Approx. 10 lines/s (continuous feed)

Figure 15. Paper Feed Stepper Motor Predetermined Time Constants

D. Relationship Between PTS and PT

Figure 13 illustrates how PTS lags PT at the start of acceleration, and moves to lead PT as the motor achieves constant speed. Figure 13 also illustrates the relationship between HR, PTS, PT, acceleration, constant speed, and printing. Figure 16 and 17 illustrate the relationship between PTS and PT during acceleration and at constant speed.

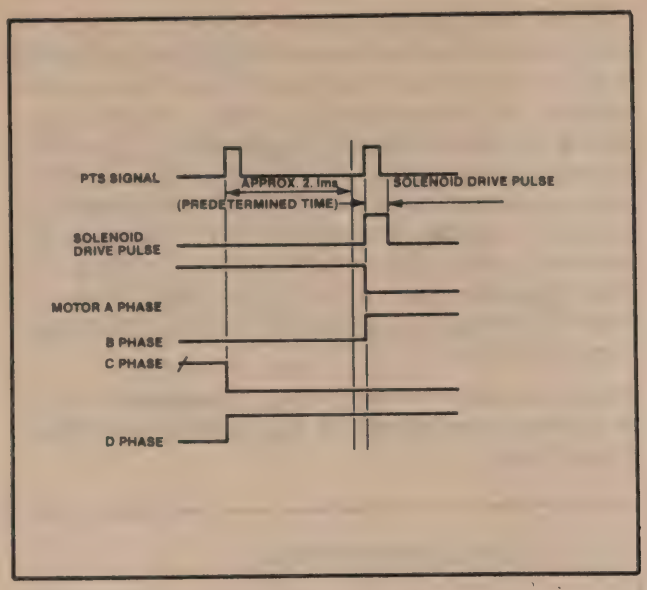


Figure 16. PTS Lags PT Timing

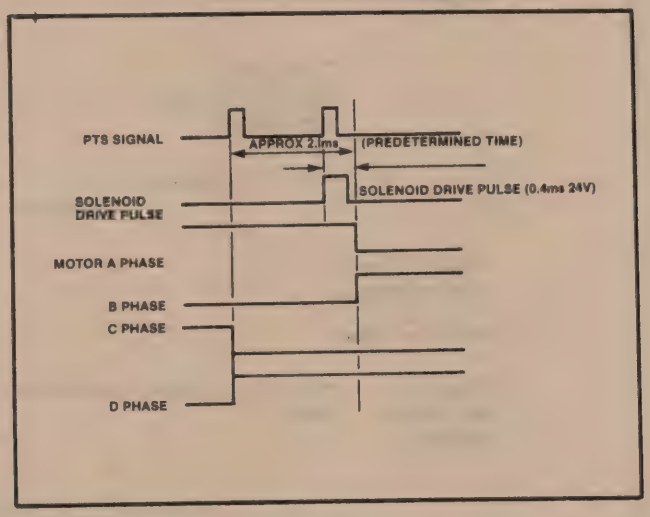


Figure 17. PTS Leads PT Timing

PTS is the point of peek angular velocity within a step of the motor. PTS is a function of the slot spacing on the encoder disk, shown in Figure 5. The spacing is determined by the mechanics of the printer mechanism.

When the carriage stepper motor is accelerated from a fixed position, the effects of damping slows the angular velocity of engergizing the stepper motor. This causes PTS to occur after the PT, or PTS lags PT. When PTS lags PT, the next step signal is output at PTS rather than at PT. If the step signal is outputted at PTS, the rotor could be midway through a rotation. Energizing the motor at PT could cause it to bind or shift in the wrong direction. When the carriage stepper motor is at a constant rate of speed, PTS leads PT and the step signal is output at PT (see Figure 13).G. Stored Time Constants.

The time between each step, for a constant number of steps, required for the motor to reach a constant speed, is calculated and stored in Data Memory during acceleration. The values stored are used, in reverse order, during deceleration as the Predetermined Time (PT) Constants. This ensures that the acceleration and deceleration distance traveled by the print head assembly is the same, and that it accurately aligns character dot columns from one line to the next during printing. The time values stored are called "Stored Time Constants." Steps T1 through T11 in Figure 13, represent the Stored Time Constants.

The equations for the Stored Time Constants are given at the bottom of Figure 13, Carriage Stepper Motor Drive Timing.

H. Print Head Assembly "Home" Position

The "logical" Home position for the print head assembly is the left-most position at which printing begins (for L-to-R motion) or ends (for R-to-L motion). The "physical" Home position is the logical HOME position, plus the distance required by the carriage stepper motor to fully accelerate the print head assembly to a constant speed. Printing can only occur when the print head is moving at a constant speed. The printer mechanism manual stipulates eleven step time periods are required to ensure the the print head assembly is at a constant speed. These eleven step time periods are the Stored Time Constants described above. Figure 18 illustrates the components of print head assembly line motion and character printing.

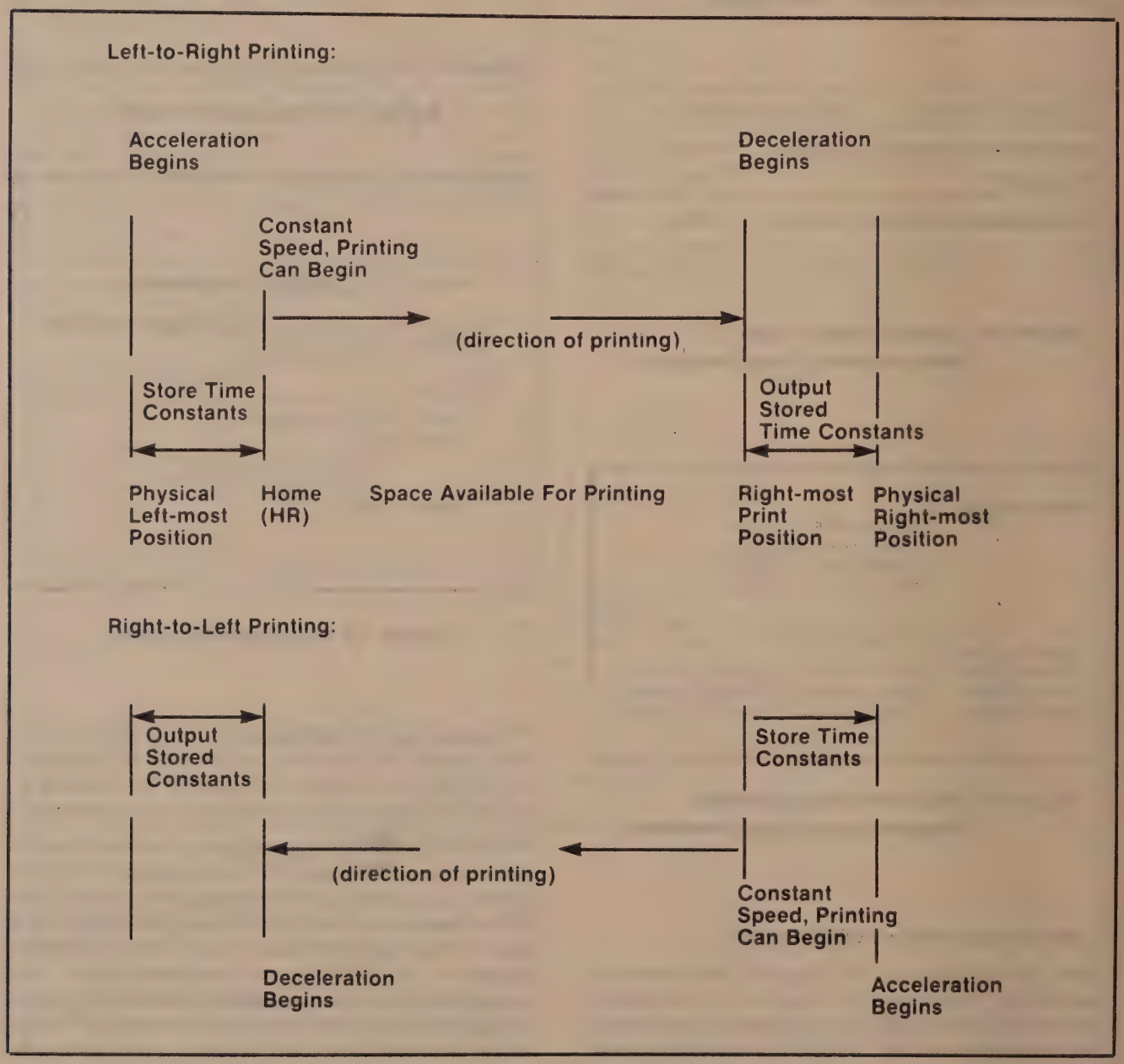


Figure 18. Components of Print Head Assembly Line Motion and Printing

SOFTWARE

Introduction

The software description is presented in three sections. First, a brief overview of the software to familiarize the reader with the interdependencies and overall program flow. Second, data and program memory allocation and status register flag definitions. And third, each of the ten software blocks is presented with its own flowchart.

Software Overview

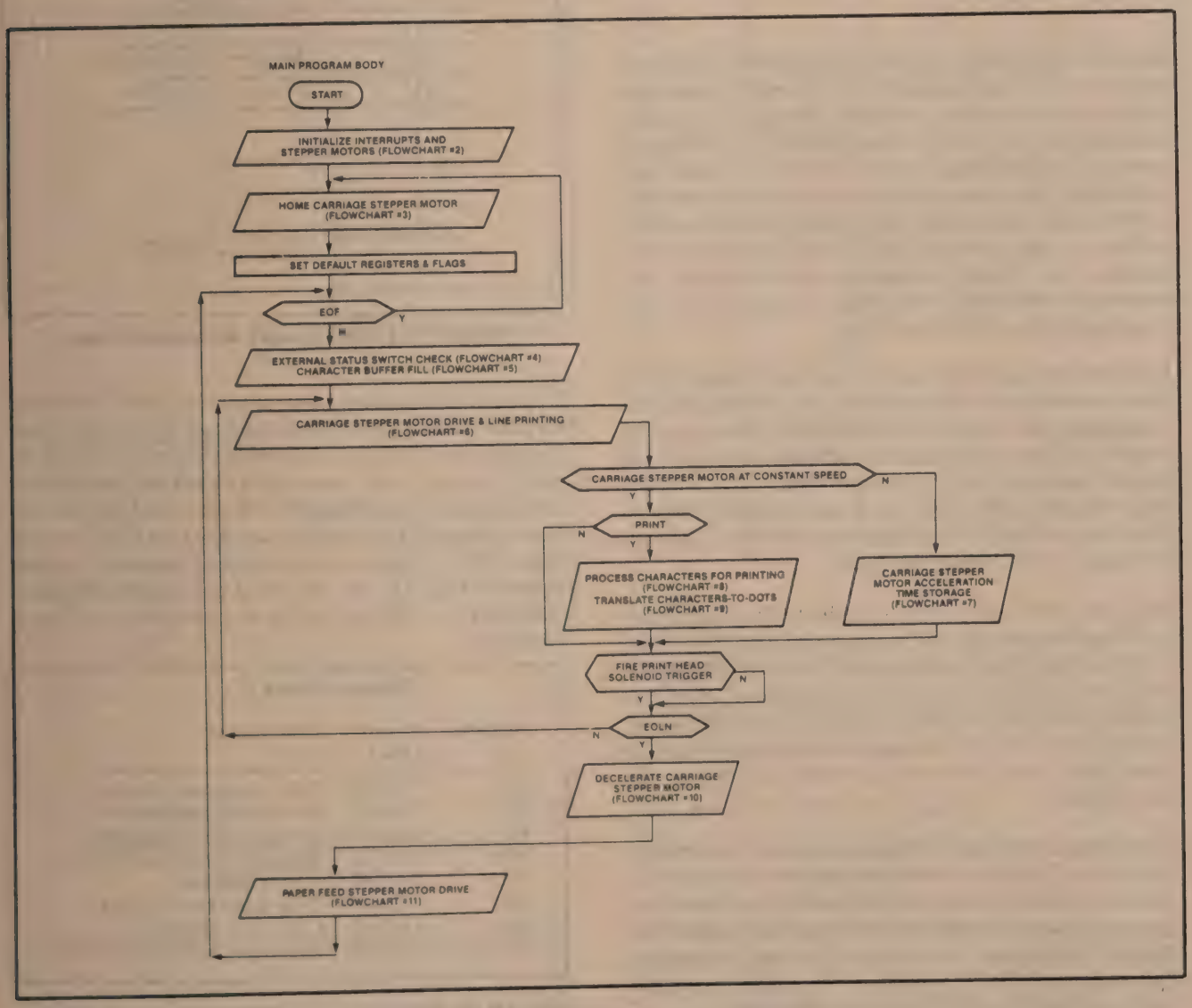
The software is written in Intel UPI-41A/42 Assembly Language. A block structure approach is used for ease of development, maintance, and comprehension. The software is divided into five principal parts.

1. Initialization
2. Character Buffering or Input
3. Stepper Motor Drive and Control
4. Character Processing
5. Character Printing or Output

The five principal parts are incorporated into ten software blocks, listed below.

1. Power On/Reset Initialization
2. Home Print Head Assembly
3. External Status Switch Check
4. Character Buffer Fill
5. Carriage Stepper Motor Drive and Line Printing
6. Accelerate Stepper Motor Time Storage
7. Process Characters for Printing
8. Translate Character-to-Dots
9. Decelerate Carriage Stepper Motor
10. Paperfeed Stepper Motor Drive

Flow Chart No. 1 illustrates the overall software algorithm. Below, is a description of the algorithm.



Flow Chart No. 1. Main Program Body

Upon power-on or reset, a software and hardware initialization is performed. This stablizes and sets inactive the printer hardware and electronics. The print head assembly is then moved to establish its HOME position. The default status registers are set for character buffering, carriage, and paper feed stepper motor drive. The External Status switches are checked; FORMFEED, LINEFEED, ON/OFF LINE, and Character Print TEST. If the printer is ON LINE, the software will loop on filling the Data Memory Character Buffer.

Character or data input to the UPI-42 is interrupt driven. Characters sent by the host system set the Input Buffer Full (IBF) interrupt and the IBF Program Status flag. Character input servicing (completed during the paper feed and carriage stepper motor drive end Delay subroutine) tests for various ASCII character codes, loads characters into the Character Buffer (CB), and repeats until one of several conditions sets the CB Full status flag. Once the CB Full flag is set, further character transmission by the host system is inhibited and printing can begin.

The carriage stepper motor is initialized, and drive begins for the direction indicated. The motor is accelerated to constant speed, printable character codes are translated to dot patterns and printed (if printing is enabled), and the motor is decelerated to a stop. Two timing loops guarantee both constant speed and protection (Failsafe Time) against stepper motor burn out due to high current overload. The two optical sensors, described in the Printer Mechanism section above, are constantly monitored to maintain constant speed, and trigger print head solenoid firing.

Once the line is printed and the carriage stepper motor drive routine has been completed, a Linefeed is forced. The paper feed stepper motor drive subroutine tests the number of lines to move, and energizes the paper feed stepper motor for the required distance. The lines per page default is 66; if 66 lines have been received, a Formfeed to Top-of-Next-Page is performed. The Top-Of-Page is set at Power On/Reset.

When the EOF code is received, the EOF status flag is set. When the last line has been printed, the EOF check will force the print head assembly to the HOME position. The EOF flag is tested following each Paper Feed stepper motor drive. The next entry to the External Status Check subroutine begins a loop which waits for input from either the external status switches or the host system.

The software character dot matrix used in this application is 5 x 7 of the available 9 x 9 print head solenoid matrix. Although lower case descenders and block/line graphics characters are not implemented, Appendix B, Printer Enhancements, discusses how and where these enhancements could be added. The software implements the full 95 ASCII printable characters set.

Memory and Register Allocation

Data Memory Allocation (RAM)

The UPI-42 has 128 bytes of Data Memory. Sixteen bytes are used by the two 8 byte register banks (RB0 and RB1). Sixteen additional bytes are used for the Program Stack. The Stored Time Constants utilize 11 bytes, while the stepper motor phase storage requires 4 bytes. Below is a detailed description of Data and Program Memory

Hex Address	Description

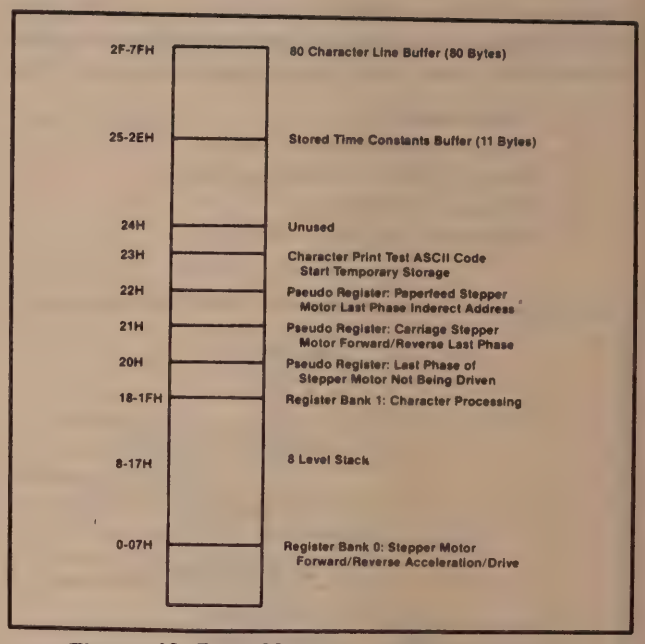


Figure 19. Data Memory Allocation Map

Register Bank 0 is used for stepper motor drive functions. Register Bank 1 is used for character processing. Each register bank's register assignments is listed in Figure 20 and 22, respectively. Each register bank has one register allocated as a Status Register. Figure 21 and 23 detail the Status Register flag assignments. Note that bit 7 of each Status Byte is used as a print head assembly motion direction flag. This saves coding of the Select Register Bank (SEL RBn) instruction at each point the flag is checked.

Register Bank 0

Register	Program Label	Description
R0	TmpR00	RB0 Temporary Register
R1	TStrR0	Store Time Register
R2	GStR20	General Status Register
R3	PhzR30	Stepper Motor Step Register
R4	CntR40	Count Register
R5	TConR0	Time Constant Register
R6	LnCtR0	Line Count Register
R7	OpnR70	Available, Scratch

Figure 20. Register Bank 0 Register Assignment

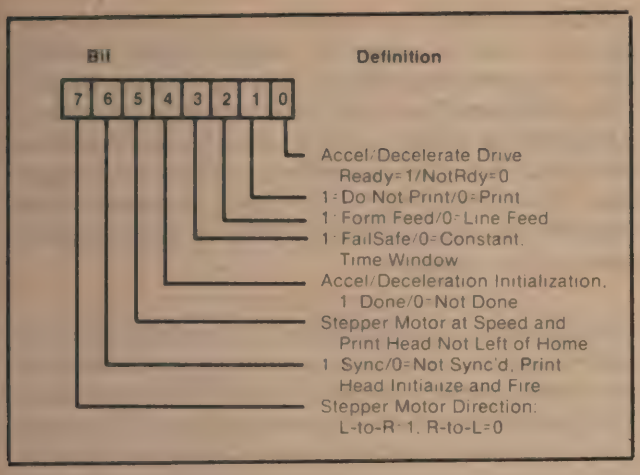


Figure 21. Register Bank 0 Status Byte Flag Assignments

Register	Program Label	Description
R0	TmpR10	RB0 Temporary Register
R1	CAdrR1	Character Data Memory Address Register
R2	ChStR1	Character Processing Status Byte Register
R3	CDtCR1	Character Dot Count Register
R4	CDotR1	Character Dot Temporary Storage Register
R5	CCntR1	Character Count Temporary Register
R6	StrCR1	Store Character Register
R7	OpnR71	Available/Scratch

Figure 22. Register Bank 1 Register Assignment

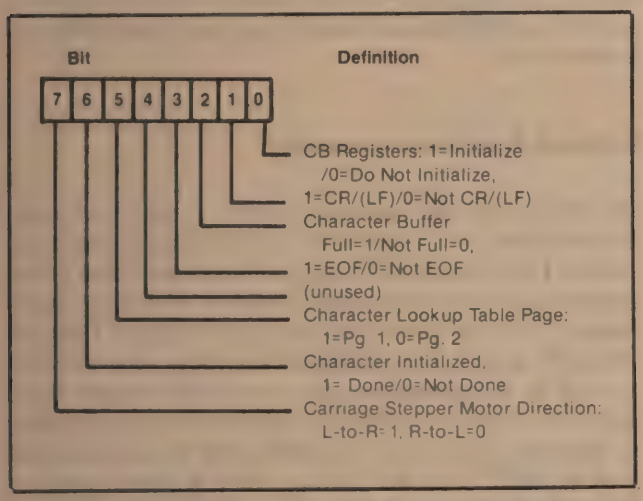


Figure 23. Register Bank 1 Status Byte Flag Assignments

Program Memory Allocation (EPROM/ROM)

The UPI-42 has 2048 bytes of Program Memory divided into eight pages, each 256 bytes. Figure 24 illustrates the Program Memory allocation map by page.

Page	Hex Address	Description
Page 7	1792-2047	Character to Dot Pattern Lookup Table; Page 2: ASCII 50H-7EH
Page 6	1536-1791	Character to Dot Pattern Lookup Table; Page 1: ASCII 20H-4FH (sp-M)
Page 5	1280-1535	Miscellaneous Subroutines: InitAl/AllOff Clear Data Memory Home Print Head Assembly Character Print Test Initialize Carriage Stepper Motor Delay Stepper Motor Deselect
Page 4	1024-1279	Paper Feed Stepper Motor Drive
Page 3	768-1023	Stepper Motor Step LookUp Table(Indexed) Character Processing and Translation Print Head Firing
Page 2	51–767	Carriage Stepper Motor Acceleration Time Calculation and Storage Stepper Motor Deceleration
Page 1	256-511	Carriage Stepper Motor Drive
Page 0	0-255	Initialization - Jump-on-Reset Main Program Body External Status Switch Check Character Buffer Fill

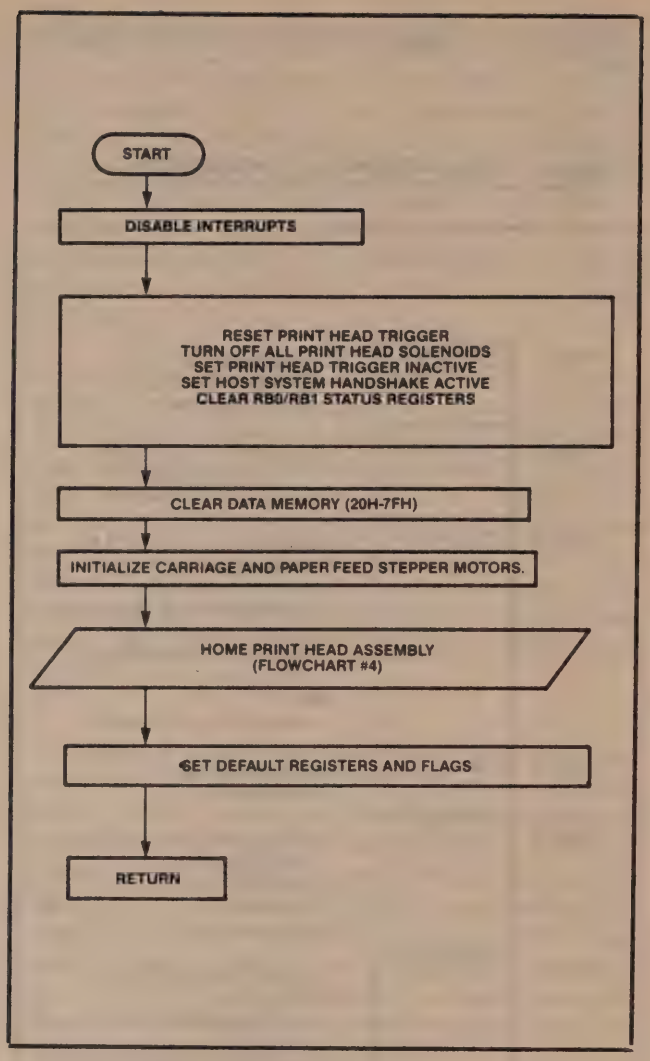
Figure 24. Program Memory Allocation Map

Software Functional Blocks

Below is a decription and flow chart for each of the ten software blocks listed above.

1. Power-On/Reset Initialization

The first operational part in Flow Chart No. 1 is the Power-On or Reset Initialization. Flowchart No. 2 illustrates the Initialization sequence in detail.



Flow Chart No. 2. Power-On/Reset Initialization

Initialization first disables both interrupts. This is done as a precaution to prevent the system software from hanging-up should an interrupt occur before the proper registers and Data Memory values are initialized.

Initialization then deactivates the system electronics. This is also a precaution to protect the printer mechanism and includes the print head solenoid (trigger and data) lines and the stepper motor select lines. The host system handshake signals are activated to inhibit data transfer from the host until the printer is ready to accept data.

Next, Data Memory is cleared from 20H to 7FH. This includes; the 80 byte Character Buffer, the 11 byte Stored Time Constants buffer, and the 4 bytes used as pseudo registers. The pseudo registers are Data Memory locations used as if they were registers. They serve as storage loacations for step data used in accurately reversing the direction of the carriage stepper motor, and stablizing either of the stepper motors not being driven.

The Data Memory locations 00H through 1FH are not cleared. These locations are Register Bank 0 (00H-07H), Program Stack (08H-17H), and Register Bank 1 (18H-1FH) (see Figure 19). Clearing the Program Registers or Stack would cause the initialization subroutine to become lost. The registers are used from the beginning of the program. Care is taken to initialize the registers and stack accurately prior to each program subroutine as required.

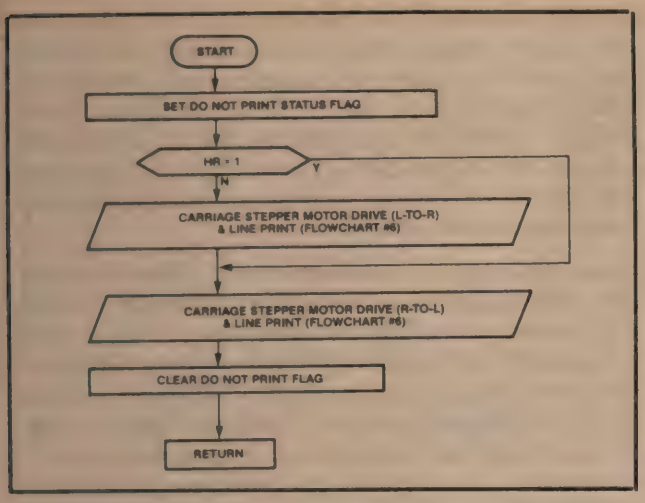
Upon power-on, it is necessary to initialize the two stepper motors, verify their operation, and locate the print head assembly in the left-most 'HOME' position. This sequence serves as a system checkout. If a failure occurs, the motors are deselected and the external status light is turned on. Each stepper motor is selected and energized for a sequence of four steps. This serves to align and stabilize each stepper motor's rotor position, preventing the rotor from skipping or binding when the first drive sequence begins.

At the end of each stepper motor's initialization, the last step data address is stored in one of the Data Memory pseudo registers. The last step data address is recalled at the beginning of the next corresponding stepper motor drive sequence, and used as the basis of the next step sequence. This ensures that the stepper motor always receives the exact next step data, in sequence, to garantee smooth stepper motor motion. This also garantees the motor never skips or jerks, which would misalign the start, stop, and character dot column positions. A stepper motor not being driven has its last phase data output held constant to stabilize it.

Following any stepper motor drive sequence of either motor, a delay of 30-60 ms occurs by switching the current to Hold Current, stabilizing the motor before it is deselected.

2. Home Print Head Assembly

At the end of the carriage stepper motor four step initialization, the output of the HR optical sensor is tested. The level of the HR signal indicates which drive sequence will be required to 'HOME' the print head assembly. If the print head assembly is to the right of HR, HR is high, the print head assembly need only be moved to from Right-to-Left until HR is low, then decelerated to locate the physical home position. If HR is low, the print head assembly must be moved first Left-to-Right until HR is high, then Right-to-Left to locate both the logical and physical 'HOME' positions. In each case, the software accelerates the carriage stepper motor, generating the Stored Time Constants then decelerates the stepper motor using the Stored Time Constants (see Background section above). Flow Chart No. 3 details the HOME print head assembly subroutine. Figures 13 and 18 illustrate the components of acceleration and print head assembly line motion.



Flow Chart No. 3. HOME Print Head Assembly

The carriage stepper motor drive subroutines used to HOME the print head assembly and to print, are the same. A status flag, called Do-Not-Print, determines whether the Character Processing subroutine is called. The flag is set by the subroutine which calls the Carriage Stepper Motor Drive subroutine. Details of the carriage and paper feed stepper motor drive and character processing subroutines are covered separately below.

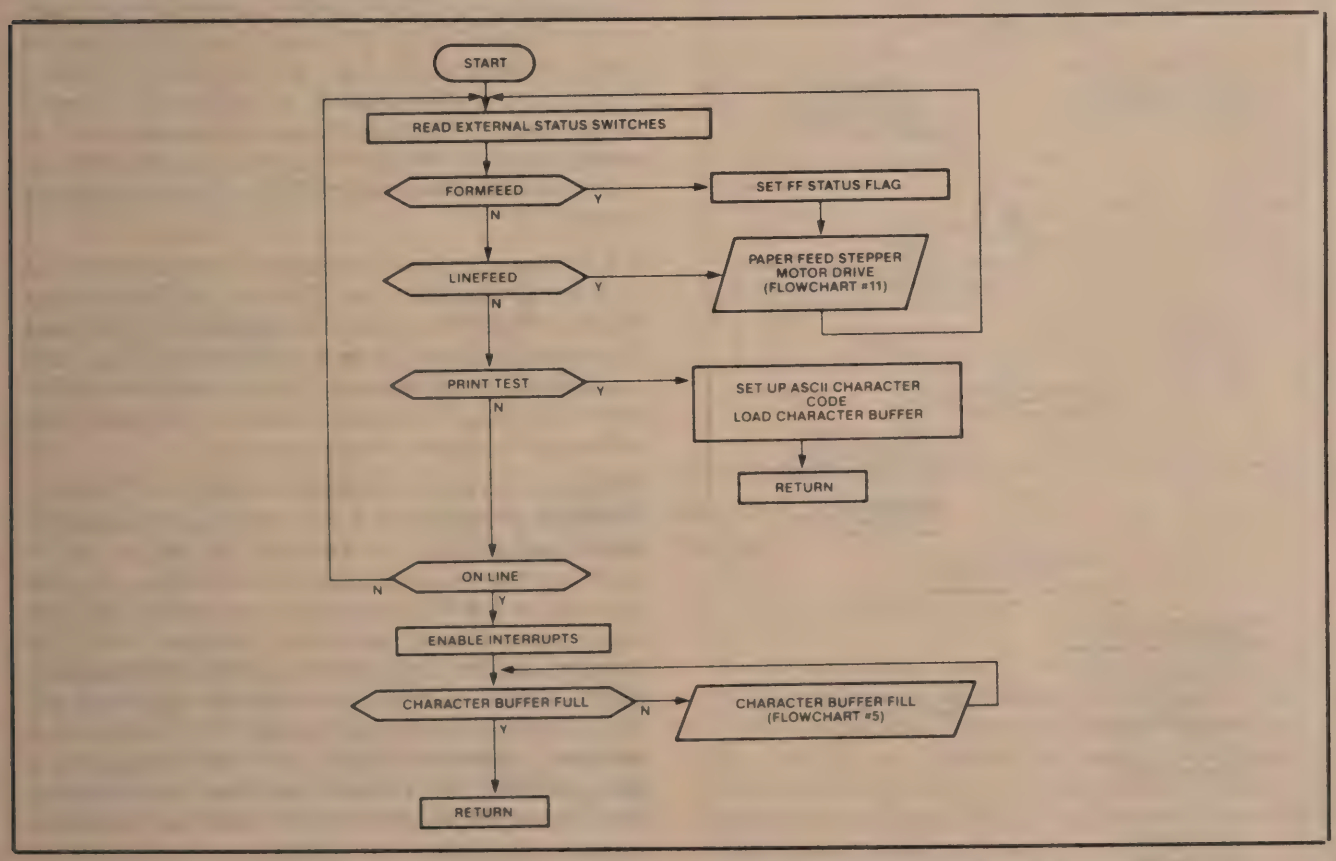
3. External Status Switch Check

Once the system is initialized and the print head is at the HOME position, the software enters a loop which continually monitors the four external status switches, and exits if any one is active. Flow Chart No. 4 details the External Status Switch Check subroutine.

Flow Chart No. 4. External Status Switch Check

If the LINEFEED or FORMFEED switch is set, the Paper Feed subroutine is called. The Paper Feed subroutine is discussed in detail below. If the ONLINE switch is set, the Character Buffer (CB) Fill subroutine is called.

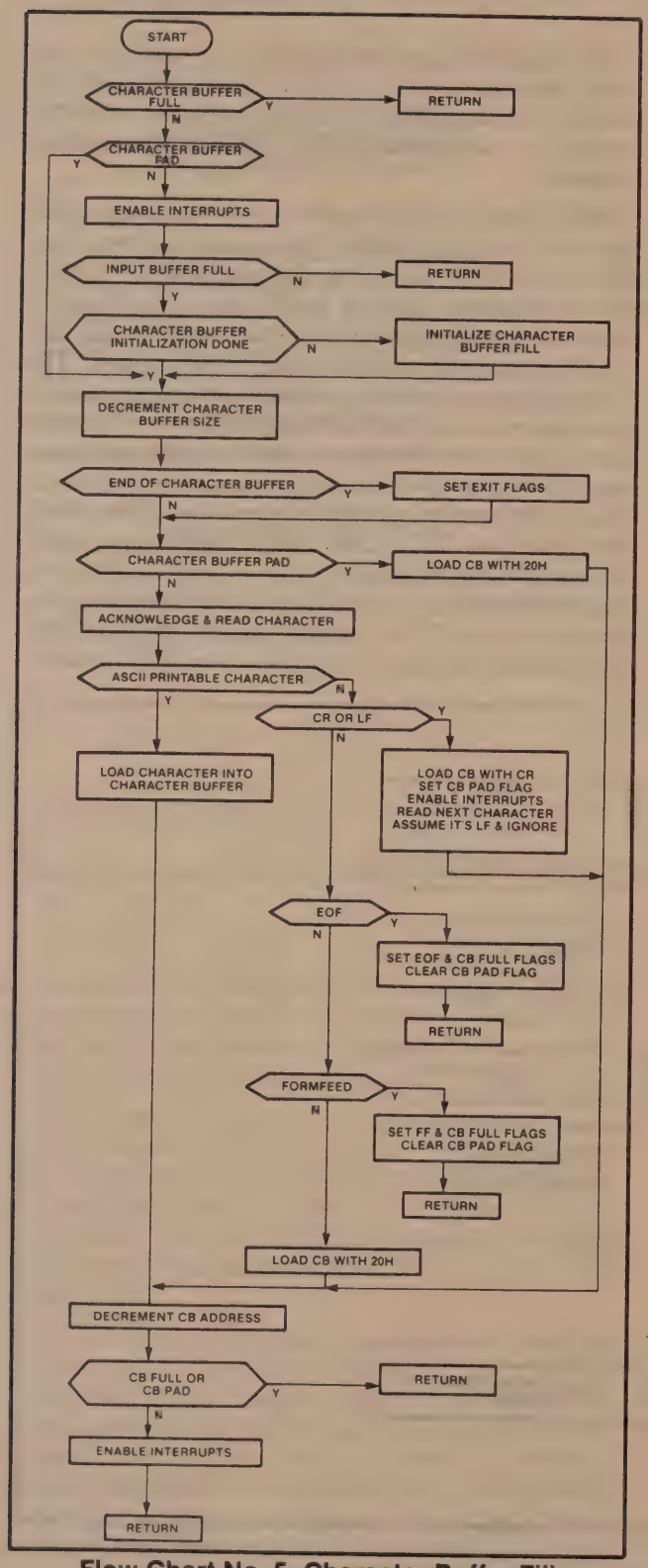
If the Character Print TEST switch is set, the Data Memory Character Buffer (CB) is automatically loaded with the ASCII code sequence, beginning at 20H (a Space character), the first ASCII printable character code. The software then proceeds as if the CB had been filled by characters received from the host system. The External Status Switch Check subroutine is exited and character printing begins. When the line has finished printing, a linefeed occurs (as shown in the main program Flow Chart No.1) and the program returns to the External Status Switch Check subroutine. If the TEST switch remains active, the ASCII character code is incremented and program continues as before. This will eventually print all 95 ASCII printable characters. An example of the TEST printer output, the complete ASCII character code printed, is shown in Figure 25.



Flow Chart No. 4. External Status Switch Check

4. Caracter Buffer Fill

The Character Buffer (CB) Fill subroutine is called from three points within the main program; External Status Switch subroutine, and the Delay subroutine following the carriage and paper feed stepper motor drive subroutines. Flowchart No. 5 details the Character Buffer Fill subroutine operation.



Flow Chart No. 5. Character Buffer Fill

The approximate 80 ms total pre-deselect delay at the end of each stepper motor drive sequence, 40 ms carriage and 40 ms paper feed stepper motor pre-deselect delay, is sufficient to load an entire 80 character line. Half the CB is filled at the end of printing the current line, and the second half is filled at the end of a paper feed. There is no time lost in printing throughput due to filling the character buffer.

Character input is interrupt driven. When the IBF interrupt is enabled, a transmitted character sets the IBF interrupt and IBF Program Status flag. Three instructions make up the IBF interrupt service routine. This short routine disables further interrupts, sets the BUSY handshake line active, inhibiting further transmission by the host, and returns. The subroutine can be executed at virtually any point in the software flow without effecting the printer mechanism operation. Processing of the received character takes place during one of the three program segments mentioned above. The BUSY line remains active until the character is processed by the CB Fill subroutine.

The CB is 80 bytes from the top of Data Memory (30H-7FH). It is a FIFO for forward, left-to-right printing, and a LIFO for reverse, right-to-left, printing. Loading the CB always begins at the top, 7FH. One character may be loaded into the buffer each time the CB Fill subroutine is called.

The CB is always filled with 80 bytes of data prior to printing. If the total number of characters input up to a Carriage Return (CR)/Linefeed (LF), does not completely fill the CB, the CR code is loaded into the CB and the balance of the CB is padded with 20H (Space Character) until the CB is full. A Linefeed (LF) character following a Carriage Return is ignored. A LF is always forced at the end of a printed line. When the CB is full, the CB Full status byte flag is set and printing can begin.

A LF character alone is treated as a CR/LF at the end of a full 80 character line. This is a special case of a printed line and is handled during character processing for printing (see No. 7, Processing Characters for Printing, below). A Formfeed (FF) character sets the FF status byte flag. The flag is tested at each paper feed stepper motor drive subroutine entry.

When the software is available to load the CB with a character, entry to the CB Fill subroutine checks three status flags; CB Full, CB Pad, and IBF flag. If the CB Full flag is set, the program returns without entering the body of the CB Fill subroutine. The CB Pad flag will cause another Space character to be loaded. If the IBF flag is not set, the program returns. If the IBF flag is set, the character is read from the Data Bus Buffer register, tested for printable or nonprintable ASCII code, and, if printable, loaded into the CB. If the character is a non-printable ASCII code and not an acceptable ASCII control code (CR, LF, FF, EOF), a 20H (Space Character) is loaded into the CB.

Exiting the CB Full subroutine with the CB Full or CB

```
]^_`abcdefghijklm
]^_`abcdefghijklmn
]^_`abcdefghijklmno
]^_`abcdefghijklmnop
]^_`abcdefghijklmnopq
]^_`abcdefghijklmnopqr
]^_`abcdefghijklmnopqrs
]^_`abcdefghijklmnopqrstu
]^_`abcdefghijklmnopqrstuv
]^_`abcdefghijklmnopqrstuvw
]^_`abcdefghijklmnopqrstuvwx
]^_`abcdefghijklmnopqrstuvwxy
]^_`abcdefghijklmnopqrstuvwxyz
]^_`abcdefghijklmnopqrstuvwxyz{
]^_`abcdefghijklmnopqrstuvwxyz{|
]^_`abcdefghijklmnopqrstuvwxyz{|}
]^_`abcdefghijklmnopqrstuvwxyz{|}~
]^_`abcdefghijklmnopqrstuvwxyz{|}~
]^_`abcdefghijklmnopqrstuvwxyz{|}~ !
]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"
]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#
]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"##
]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%
]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%&
]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%&'
]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%&'(
]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%&'()
]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%&'(*
]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%&'(*+
]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%&'(*+,
]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%&'(*+,-
]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%&'(*+,-.
]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%&'(*+,-./
]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%&'(*+,-./0
]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%&'(*+,-./01
]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%&'(*+,-./012
]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%&'(*+,-./0123
]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%&'(*+,-./01234
]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%&'(*+,-./012345
]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%&'(*+,-./0123456
]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%&'(*+,-./01234567
]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%&'(*+,-./012345678
]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%&'(*+,-./0123456789
]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%&'(*+,-./0123456789:
]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%&'(*+,-./0123456789:;
]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%&'(*+,-./0123456789:<
]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%&'(*+,-./0123456789:;<=
```

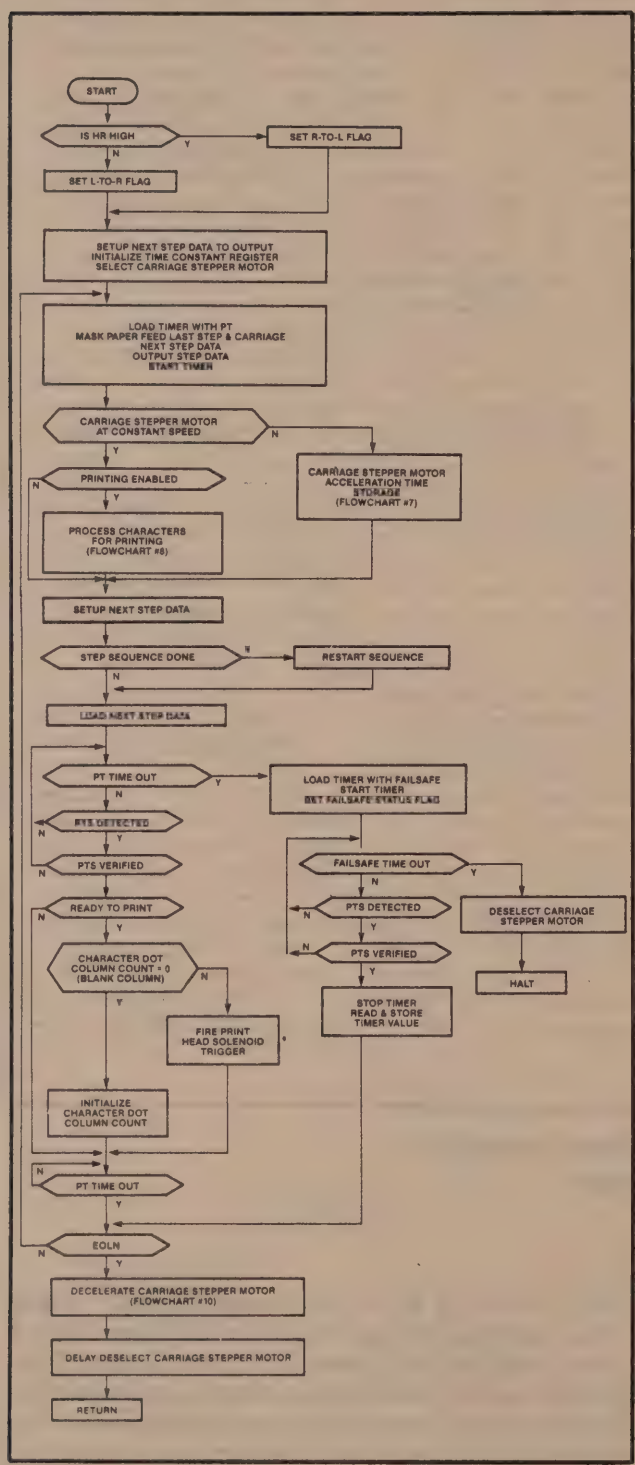
Figure 25. ASCII Character Code TEST Output and Print Example

Pad flag set does not re-enable IBF interrupts or reset the BUSY line. If neither of these flags is set, exiting the CB Fill subroutine sets BUSY inactive and IBF interrupts are enabled. Once the CB Full status byte flag is set, IBF interrupts are disabled until the CB has been entirely emptied, the line printed, or the system Reset.

5. Carriage Stepper Motor Drive and Line Printing

The carriage stepper motor drive subroutine controls both L-to-R and R-to-L print head assembly motion. Upon entering the subroutine, the HR signal level is tested to determine the direction of print head assembly motion and the Direction status flag is set. The default control register values are loaded and balance of the default status flags are set for stepper motor control and character processing. The default control register values include PT and the step sequence look-up table start address for the direction indicated.

The direction flag is tested throughout the carriage stepper motor drive and character processing subroutines. This enables the same subroutines to control activities for either direction, simplifying and shorting the overall program. Flow Chart No. 6 illustrates the carriage stepper motor drive subroutine.



Flow Chart No. 6. Carriage Stepper Motor Drive/Line Printing

Next, the carriage and paper feed stepper motor step data is initialized. The last step data output to the paper feed stepper motor is loaded into the Last Phase pseudo register. This data is masked with each step data output to the carriage stepper motor. Masking the step data in this manner guarantees the paper feed motor signals do not change as the carriage stepper motor is being driven.

Figure 26 illustrates the carriage stepper motor step sequence verses the actual step data output for clockwise rotation, Left-to-Right motion, and counterclockwise rotation, Right-to-Left print head assembly motion. An eight step sequence is depicted in the figure. Note that the sequence for Right-to-Left motion is the reverse of the sequence for Left-to-Right motion. Note also, that for the L-to-R sequence step 4 is the same as step 0, step 5 the same as step 1, etc., through step 7 matching step 3. The four step sequence simply repeats itself until the motor is stopped via the Deceleration subroutine.

L-to-R Motion Sequence	Phase/Step Data (3 2 1 0)	R-to-L Motion Sequence	BCD (3 2 1 0)
0	1 0 0 1	7	0 0 0 0
1	1 0 1 0	6	0 0 0 1
2	0 1 1 0	5	0 0 1 0
3	0 1 0 1	4	0 0 1 1
4	1 0 0 1	3	0 1 0 0
5	1 0 1 0	2	0 1 0 1
6	0 1 1 0	1	0 1 1 0
7	0 1 0 1	0	0 1 1 1

Figure 26. Carriage Stepper Motor Phase/Step Data

When the carriage stepper motor is driven for a specific direction of print head assembly motion, the step sequence must be consistant for the motion to be smooth and accurate. The same holds true for the transition from one direction of motion to the other. Since the sequence for one direction is the opposite for the other direction, incrementing the sequence for L-to-R and decrementing for R-to-L provides the needed step data flow. For example, referring to Figure 26, if the print head assembly moved L-to-R and the last step output was #1, the first step for R-to-L motion would be #7. Thus, when the carriage stepper motor is initialized for a clockwise (L-to-R) or counterclockwise (R-to-L) rotation, the last step sequence number is incremented or decremented to obtain the proper next step. In this way, the smooth motion of the stepper motors is assured.

The step data is referenced indirectly via the step sequence number. The step data is stored in a Program Memory look-up table whose addresses correspond to the step sequence numbers. For example, as shown in

Figure 26, at location 0 the step data "1001" is stored. This method is particularly well suited to the UPI-42 software. The UPI-42 features a number of instructions which perform an indirect move or data handling operation. One of these instructions, MOVP3 A,@A, unlike the others, allows data to be moved from Page 3 of Program Memory to any other page of Program Memory. This instruction allows the step data to be centrally located on Page 3 of Program Memory and accessed by various subroutines.

Each time the carriage stepper motor step data is output, the step data lookup table address is incremented or decremented, depending upon the direction of rotation, and tested for restart of the sequence. The address is tested because the actual step data, Figure 26, is not a linear sequence and thus is not an easily testable condition for restarting the sequence. The sequence number is tested for rollover of the sequence count from 03H to 04H and clockwise motor rotation via the Jump on Accumulator Bit instruction (JBn), with 00H loaded to restart the sequence. The same bit is tested when decrementing the sequence count for counterclockwise motor rotation, R-to-L motion, because the count rolls over from 00H to 0FFH, with 03H loaded to restart the sequence.

At this point the UPI-42 Timer/Counter is loaded, the step signal is output, and the timer started. The next step data to be output has been determined and the At-Speed flag is tested for entry to one of two subroutines; Stepper Motor Acceleration Time Storage or Character Processing.

The first entry to the Acceleration Time Storage subroutine initializes the subroutine and returns. All other entries to one of the two subroutines perform the necessary operations, detailed below (Blocks 6 and 7), and returns. The program loops until the PT times out or the PTS level change is detected. PTS is tied to T0 of the UPI-42. The level present on T0 is directly tested via conditional jump instrunctions. The software loops on polling the timer Time Out Program Status flag and the T0 input level.

As described in the Background section above (shown in Figure 13), if PT times out before PTS is detected, the software waits for PTS before outputing the next step signal. If PT times out before PTS, a second timer count value is loaded into the UPI-42 timer. The timer value is called "Failsafe." This is the maximum time the stepper motor can be selected, with no rotor motion, and not damage the motor. If PTS is not detected, either the carriage stepper motor is not rotating or the optical sensor is defective. In either case, program excution halts, the motor is deselected, and the external status light is turned on to indicate a malfunction. A system reset is required to recover from this condition. The Failsafe time is approximately 20 milliseconds, including PT.

The Failsafe time loop also serves as a means of tracking the elapsed time between PT time out and PTS. Entry to the Failsafe time loop sets the Failsafe/Constant Time Window status flag. This flag is tested by the Acceleration Time Storage subroutine for branching to the proper time storage calculation to be perform (see Figure 13 and Block 6 below for further description).

During the Failsafe timer loop, if PTS is detected and verified as true, the Failsafe timer value is read and stored in the Time Storage register. This value is used during the next Acceleration Time Storage subroutine call to calculate the Stored Time Constant (see Block 6 below). If PTS is invalid, the flow returns to the timer loop just exited, again waiting for PTS or Failsafe time out.

During the PT time loop, if PTS is detected and verified, the Sync flag is tested for entry to the print head solenoid firing subroutine. This flag is set by the first entry to the Character Processing subroutine. The flag synchronizes the solenoid firing with character processing. Only if characters are processed for printing will the solenoids be enabled, via the Snyc flag, for firing. This prevents the solenoids from being fired without valid character dot data present.

As described in the Background section "Relationship Between PTS and PT," PTS is the point of peek angular velocity within a step of the motor. After PTS is detected the motor speed ramps down, compensating for the overshoot of the rotor motion. PTS is the optimum time for print head solenoid firing, as shown in Figure 13. This is the most stable point of motor rotation and, thus, the print head assembly motion. If PTS is detected during PT, printing is enabled, the Sync flag is set, and the solenoid trigger is fired.

The firing of the solenoid trigger, following PTS, is very time critical. The time between PTS and solenoid firing must be consistant for accurate dot column alignment throughout the printed line. The software is designed to meet this requirement by placing all character processing and motor control overhead before the solenoid firing subroutine is called. The actual instruction sequence which fires the print head solenoid trigger is plus or minus one instruction for any call to the subroutine.

Once the timer loop is complete, the software tests for Exit conditions. If the Exit conditions fail, the software loops to output the next step signal, starts the PT timer, and continues to accelerate the carriage stepper motor, or process, and print characters. If the Exit test is true, the carriage stepper motor is decelerated to a fixed position, and the program returns to the main program flow (see Flowchart 1).

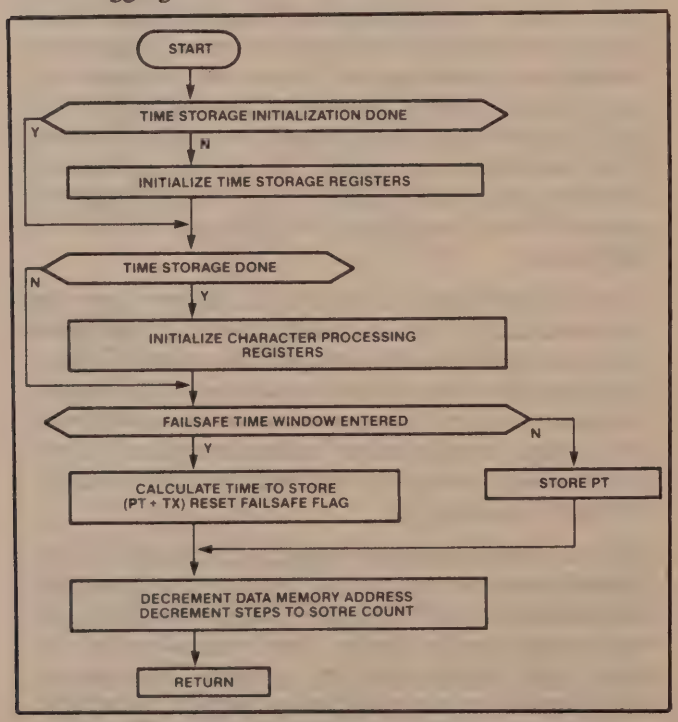
The exit conditions are different for the two directions of print head assembly motion. For L-to-R printing, if a Carriage Return (CR) character code is read from CB, the carriage stepper motor drive terminates and the motor is decelerated to a fixed position. There are two conditions for terminating carriage stepper motor drive upon detecting a CR during L-to-R motion. If less than half a character line (40 characters) has been printed,

the print head assembly returns to the HOME position to start the next printed line. Otherwise, the print head assembly continues to the right-most position for a full 80 character line, and then begins printing the next line from R-to-L. R-to-L printing always returns the print head assembly to the HOME position before the next line is printed L-to-R. When HR is high, character printing always stops and the carriage stepper motor drive subroutine exits to the deceleration subroutine.

6. Accelerate Stepper Motor Time Storage

As described above, when the carriage stepper motor is accelerated the step time required to guarantee the motor is at a constant rate of speed translates to a specific distance traveled by the print head assembly (see Figure 18). In order to position the print head assembly accurately for bi-directional printing, the distance traveled during deceleration must be the same as during acceleration. The Carriage Motor Acceleration Time Storage subroutine calculates the step times needed to accelerate the carriage stepper motor, and stores them in Data Memory for use as PT during deceleration.

The first call of the Carriage Stepper Motor Acceleration Time Storage subroutine initializes the required registers and status flags. The time calculation begins with the second carriage stepper motor step signal output. The program returns to the carriage stepper motor drive subroutine and loops on PT. Each subsequent call of the Acceleration Time Storage subroutine tests the Failsafe/Constant flag and branches accordingly (see Flow Chart 7). The Acceleration Time Storage subroutine has two parts which correspond to PTS leading or PTS lagging PT.



Flow Chart No. 7. Carriage Stepper Motor Acceleration Time Storage

If the Failsafe/Constant flag is set, PTS lagged PT. The time from PT time out to PTS, Tx (see Figure 13), must be added to the PT and stored in Data Memory. As described above, if PT lagged PT, the Failsafe time is loaded and PTS is again polled during the time loop. When PTS occurs within the Failsafe time, the timer is stopped and the timer value stored. The UPI-42 timer is an up timer, which means that the value stored is the time remaining of the Failsafe time when PTS occured. The elapsed time must be calculated by subtracting the time remaining (the value stored) from the Failsafe time constant. This is done in software by using two's complement arithmetic. If the Failsafe flag is not set PTS led PT, and PT is the Stored Time Constant stored.

Indirect addressing of Data Memory is used to reference the Stored Time Constant Data Memory location. The Data Memory location address is decremented each time the Acceleration Time Storage subroutine is exited and a Stored Time Constant has been generated.

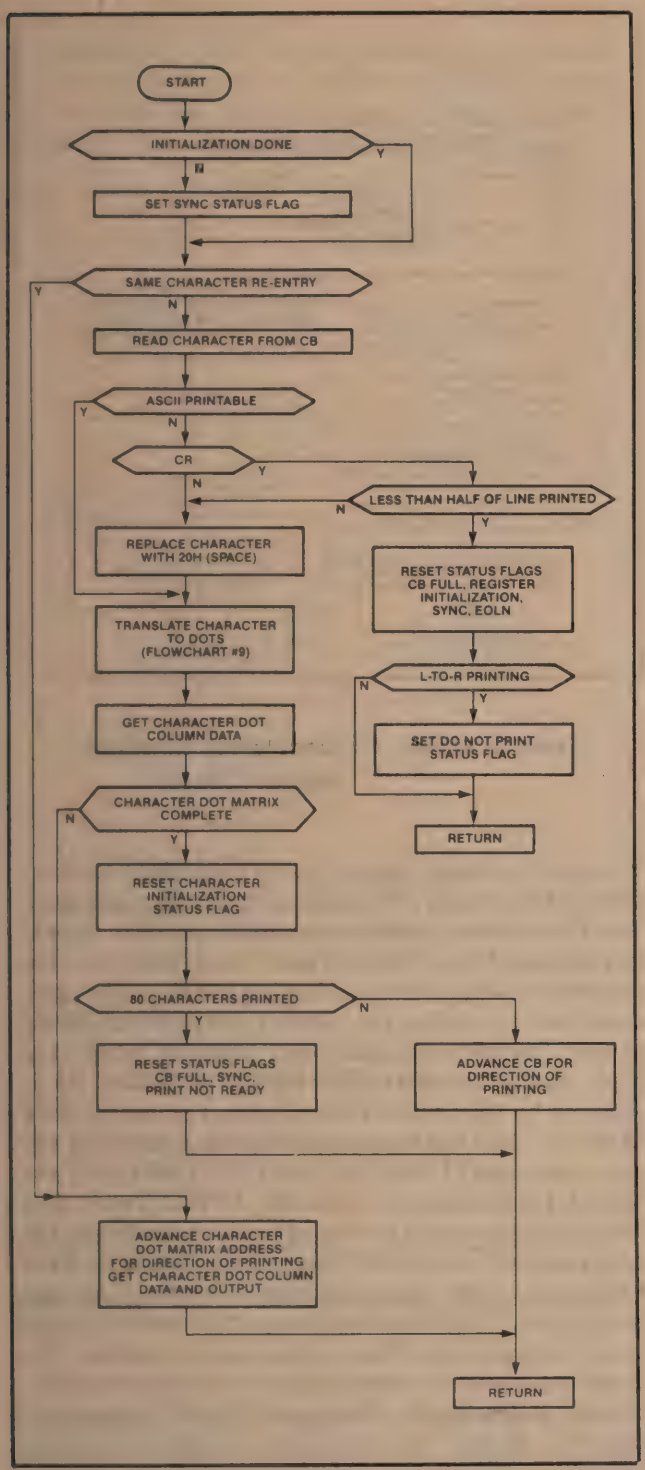
The last Acceleration Time Storage subroutine exit sets the At-Speed status flag and initializes the character processing registers and flags.

3. Process Characters for Printing

The Character Processing subroutine is entered only if the Home Reset (HR) optical sensor signal is high and printing is enabled. Otherwise, the software simply returns to the Carriage Stepper Motor Drive subroutine. There are two cases when printing is not enabled; during the HOME subroutine operation, and when the print head assembly returns to the HOME position after printing less than half an 80 character line. If printing is enabled, the Sync status flag is set.

All character processing operations use the second UPI-42 Data Memory Register Bank, RB1. Register Bank 1 is independent of Data Memory Register Bank 0, used for stepper motor control. The use of two independent register banks greatly simplfies the software flow, and helps to ensure the accuracy of event sequences that must be handled in parallel. Each register bank must be initialized only once for any entry to either the Carriage Stepper Motor Drive or Character Processing subroutines. A single UPI-42 Assembly Language instruction selects the appropriate register bank. Initializing the character processing registers includes loading the maximum character count (80), dot matrix size count (6), and CB start address. The CB start address is print direction dependant, as described in Block 4, above.

Character processing reads a character from the CB, tests for control codes, translates the character to dots, and conditionally exits, returning to the Carriage Stepper Motor Drive subroutine. Flow Chart 8 details the character processing subroutine.



Flow Chart No. 8. Process Characters for Printing

Each character requires six steps of the carriage stepper motor to print; five for the 5 character dot columns and 1 for the blank dot column between each character. Reading a character from the CB and character-to-dot pattern translation takes place during the last character dot column, or blank column, time.

The first character line entry to the Character Processing subroutine appears to the software as if a last character dot column (blank column) had been entered. The next character, in this case the first character in the line, is translated and printing can begin. This method of intiializing the Character Processing subroutine utilizes the same software for both start-up and normal character flow. Once a character code has been translated to a dot matrix pattern starting address in the look-up table, all subsequent entries to the Character Processing subroutine simply advance the dot column data address and outputs the data.

The decision to translate the character to dots during the blank column time was an arbitary one. As was the choice of the blank column following rather than preceding the actual character dot matrix printing.

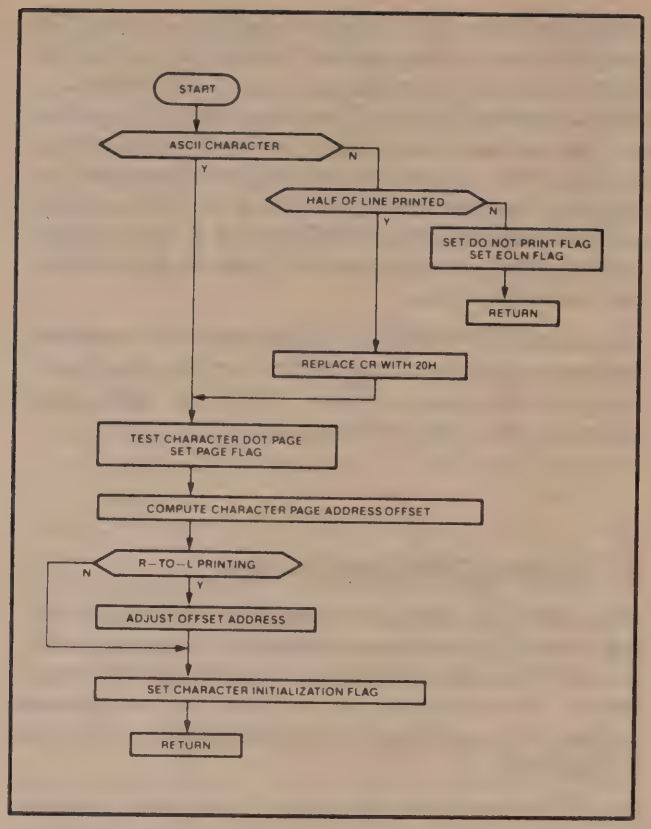
4. Translate Character-to-Dots

Character-to-dot pattern translation involves converting the ASCII code into a look-up table address, where the first of the five bytes of charcter dot column data is stored. The address is then incremented for the next column of dot pattern data until the full character has been printed.

The dot pattern look-up table occupies two pages, or approximately 512 bytes of Program Memory. A printable ASCII character is tested for its dot pattern location page and the offset address, from zero, on that page. Both the page test and page offset calculations use two's complement arithmetic, with a jump on carry or not carry causing the appropriate branching. Once the pattern page and address are determined the indirect addressing and data move instructions are used to read and output the data to the print head solenoids. Flowchart 9 details the Character-to-Dots Translation subroutine.

In the case of R-to-L printing, although the translation operation is the same, the character is printed in reverse. This requires that the character dot pattern address be incremented by five, before printing begins, so that the first dot column data output is the last dot column data of the character. The dot pattern look-up table address is then decremented rather that incremented, as in L-to-R printing, for the balance of the character. Translation still takes place during the last character dot column, the blank column, and the blank column follows the character matrix.

Only one control code, a Carriage Return (CR), is encountered by the character translation subroutine. Linefeed (LF) characters are stripped off by the CB Fill subroutine. If a CR code is detected the software tests for a mid-line exit condition; less than half the line printed exits the stepper motor drive subroutine and HOMEs the print head assembly before printing the next line. If the test fails, more than half the line has been printed, the CR is replaced by a 20H (Space character) and printing continues for the balance of the line; the space characters padding the CB are printed.



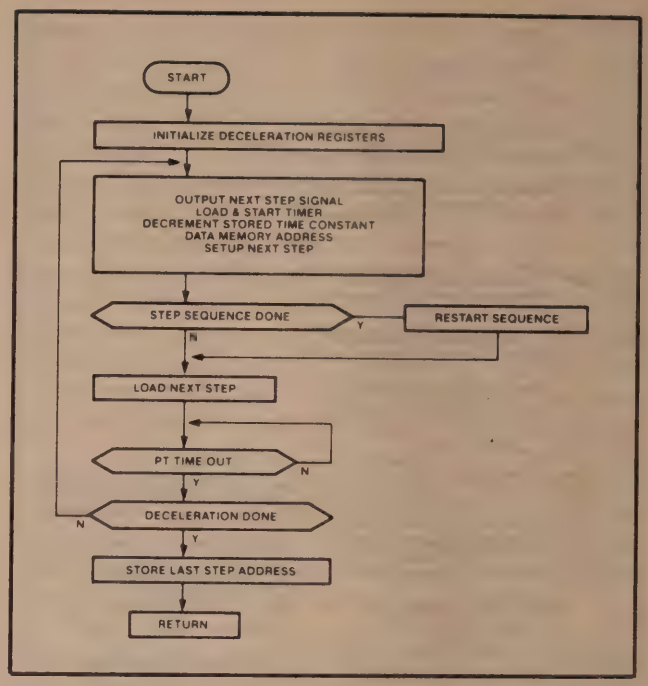
Flow Chart No. 9. Translate Character-to-Dots

As mentioned above, the character dots are printed and the print head trigger is fired when the PTS signal is detected and verified and the carriage stepper motor is At Speed.

When the character to print test fails the CB Buffer size count equals zero, the Carriage Stepper Motor Drive subroutine exit flags are set, and the flow passes to the Deceleration and Delay subroutines and programs returns to the main program flow.

9. Decelerate Carriage Stepper Motor

The transition from the Carriage Stepper Motor Drive subroutine to the Deceleration subroutine outputs the next step signal in sequence, and then initializes the Deceleration subroutine registers; Stored Time Constants Data Memory buffer end address and size. The Stored Time Constant Buffer is a LIFO for deceleration of the carriage stepper motor. The buffer size is used as the step count. When the step count decrements to zero, the step signal output is terminated, and the last step sequence number is stored in the carriage stepper motor Next Step pseudo register. The last step sequence number is recalled, during initialization of the next carriage stepper motor drive, as the basis of the next step data signal to be output. See Flow Chart 10.



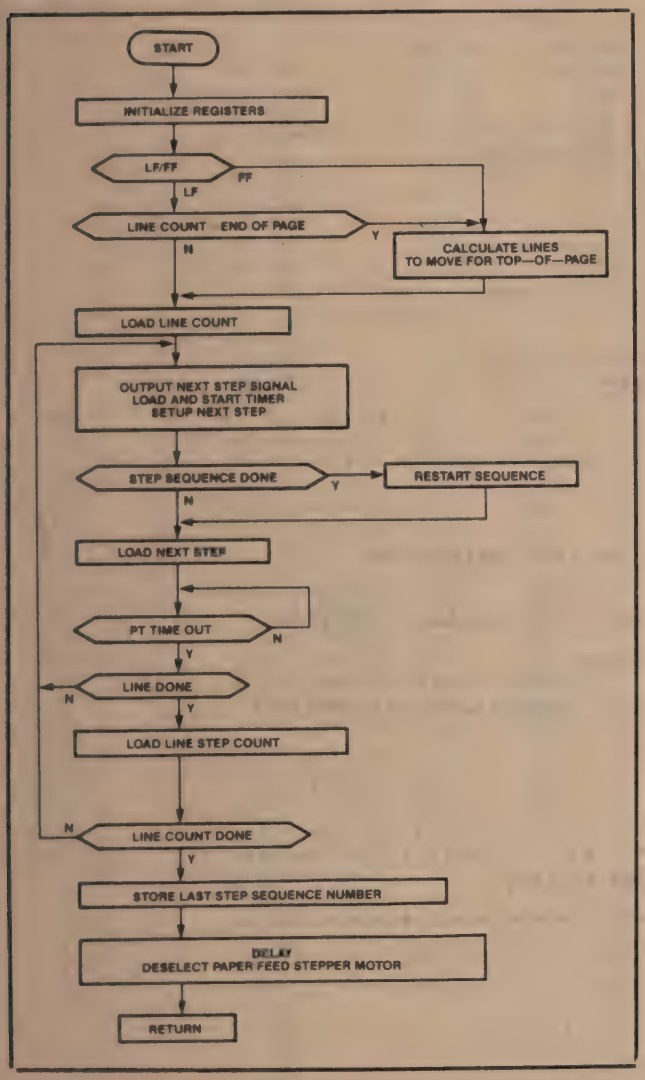
Flow Chart No. 10. Decelerate Carriage Stepper Motor

When the carriage stepper motor is decelerated, Failsafe protection and PTS monitoring are not necessary. The Deceleration subroutine acts as its own failsafe mechanism. Should the stepper motor hang-up, the subroutine would exit and deselect the motor in sufficient time to protect the motor from burnout. Since neither Failsafe nor print head solenoid firing take place during deceleration, PTS is not needed. PT is replaced by the Stored Time Constant values in Data Memory. The Deceleration subroutine determines the next step signal to output, loads the Timer with the Stored Time Constant, starts the UPI-42 Timer, and loops until time out. The subroutine loops to output the next step until all of the Stored Time Constants have been used. The program returns to the Carriage Stepper Motor Drive subroutine and the motor is deselected following the Delay subroutine execution. The Delay subroutine is called to stablize the stepper motor before it is deselected. During the DELAY subroutine, the IBF interrupt is enabled and characters are processed. A paper feed is forced following the carriage stepper motor being deselected.

10. Paper Feed Stepper Motor Drive

The paper feed stepper motor subroutine outputs a predefined number of step signals to advance the paper, in one line increments, for the required number of lines. The number of step signals per line increment is a function of the defined number of lines per inch, given the distance the paper moves in one step. Figure 16 lists three step (or pulse) count and line spacing configura-

tions, as well as the distance the paper moves in one step. Standard 6 lines per inch spacing has been implemented in this Application Note (Appendix B details how variable line spacing could be implemented). Flowchart 11 illustrates the Paper Feed subroutine.



Flow Chart No. 11. Paper Feed Stepper Motor Drive

The number of lines the paper is to be moved is called the "Line Count." The Line Count defaults to one unless the Formfeed flag is set, or the total number of lines previously moved equals a full page. The default total lines per page for this application is 66. When the total number of lines moved equals 66, the paper is moved to the top of the next page. The Top-of-Page is set at power-on or reset.

If the Formfeed flag has been set in the Character Buffer Fill subroutine, the software calculates the number of lines needed for a top of next page paper feed. The resulting line count is loaded in the Line Count Register. The Paper Feed subroutine loops on the line count until done and then returns to the main program body.

Once the Paper Feed subroutine is complete, the software loops to test the End of File (EOF) Flag (see Flow Chart 1). If EOF is set, the print head assembly is moved to the HOME position, the program again enters the External Status Switch Test subroutine, and begins polling the external status switches. If EOF is not set, the program directly calls the External Status Switch Check subroutine, and the program repeats for the next line.

CONCLUSION

Although the full speed, 12 MHz, of the UPI-42 was used, the actual speed required is approximately 8-9 MHz. 1400 bytes of the available 2K bytes of Program Memory were used; 500 bytes for the 95 character ASCII code dot pattern look-up table, 900 bytes for operational software. This means that the UPI-42 has excess processing power and memory space for implementing the additional functions such as those listed below and discussed in Appendix B.

Special Characters or Symbols
Lower Case Descenders
Inline Control Codes
Different Character Formats
Variable Line Spacing

The software developed for this Application Note was not fully optimized and could be further packed by combining functions. This would require creating another status register, which could also serve to implement some of the features listed above. Since the full 16 byte stack is not used for subroutine nesting, there are 6-8 bytes of Program Stack Data Memory that could be used for this purpose. In several places, extra code was added for clarity of the Application Note. For example, each status byte flag is set with a separate instruction, using a equate label, rather than setting several flags simultaneously at the same point in the code.

This Application Note has demonstrated that the UPI-42 is easily capable of independently controlling a complex peripheral device requiring real time event monitoring. The moderate size of the program required to implement this application attests to the effectiveness of the UPI-42 for peripheral control.

APPENDIX A. SOFTWARE LISTING

```
$MOD42  TITLE('UPI 42 APP NOTE');
$MACROFILE NOSYMBOLS NOGEN DEBUG

$INCLUDE(:F1:ANECD.OV1)
; PG

; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;                Complex Peripheral Control With the UPI-42

;                Intel Corporation
;                3065 Bowers Avenue
;                Santa Clara, Ca. 95051


;                Written By      Christopher Scott


; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

;                Notes and Comments

;       Three Assembly Language files comprise the full Application
;       Note source code;

;       1. ANECD.OV1    App Note Equates, Constants, Declarations . Overlay

;       2. 42ANC.SRC    UPI-42 App Note Code Source

;       3. CHRTBL.OV1   Character Table . Overlay  (Character Lookup Tables)


; PG
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;                Equates, Constants and System Definitions
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

;       Data & Program Memory Allocations

;       Program Memory

;       Page No.   Hex Addr      Description
; ---------------------------------------------------------------------------

;       Page 7     1792-2047    Char to Dot pattern lookup table
;                                 Page 2:  ASCII 50H-7FH (N-~)
;       Page 6     1536-1791    Char to Dot pattern lookup table
;                                 Page 1:  ASCII 20H-4FH (sp-M)
;       Page 5     1280-1535    Misc called routines:
;                                 InitAl/AllOff
;                                 Clear Data Memory
;                                 CR Home
;                                 Char Print Test -  load Ascii char codes
;                                 Initialize CR Stpr Mtr
;                                 Delay: short/long/very long
;                                 Stpr Mtr deselect
;       Page 4     1024-1279    PaperFeed Stpr Mtr Init and Drive
;       Page 3      768-1023    Stpr Mtr Phase LookUp Table - Indexed
;                               Character Translation and processing
;                               PrintHead firing
;       Page 2      512-767     Stpr Mtr Accel. Time calc. and memorization
;                               Stpr Mtr Deceleration
;       Page 1      256-511     SMDriv (FAccel/RAccel) - Forward & Reverse
;                                 Stpr Mtr acceleration & drive
;       Page 0        0-255     Initialization Jmp-on-Reset
;                               Program Body - all calls
;                               Character Input test and Char Buffer fill loop
;                               Interrupt service routines
```

```
                = 71 ; PG
                = 72 ; ------------------------------------------------------------------------
                = 73 ;      Data Memory
                = 74 ; ------------------------------------------------------------------------
                = 75 ;        Dec.        Hex          Description
                = 76 ;       -------    --------      ------------------------------------------
                = 77 ;   TOP
                = 78 ;   ---------------------
                = 79 ;       48-127     2F-7FH        80 Character Line Buffer
                = 80 ;        37-47     25-2EH        Stpr Mtr Accel/Decel time, memorization
                = 81 ;           36        24H        Unused
                = 82 ;           35        23H        Char Print test ASCII code start tmp store
                = 83 ;           34        22H        LF SM last Phz Inderect Addr psuedo reg
                = 84 ;           33        21H        CR SM Forward/Reverse last Phz psuedo reg
                = 85 ;           32        20H        Psuedo Reg:  Last Phase of stpr mtr not
                = 86 ;                                              being driven
                = 87 ;        24-31     18-1FH        Register Bank 1:  Character Handling
                = 88 ;         8-23      8-17H        8 Level Stack
                = 89 ;         0-7       0-07H        Register Bank 0:  Stpr Mtr F/R Accel/Drive
                = 90 ;   ---------------------
                = 91 ;   BOTTOM
                = 92
                = 93 ;       Data Memory Equates:
                = 94 ; ------------------------------------------------------------------------
                = 95
0050            = 96 CHBFSZ  EQU     50H            ;char buffer size 0-79 = 80
00D9            = 97 HlfCpl  Equ     0D9H           ;Cpl(1/2 CbBfSz) => cpl of 27H = 0D9H
                = 98
007F            = 99 FCBfSt  Equ     7fH            ;start of char buffer
0080            = 100 FCBfIS  Equ     80H            ;init CB strt-allows xtra Dec by 1
002F            = 101 RCBfIS  Equ     2FH            ;init CB strt-allows xtra Inc by 1
0051            = 102 ChBfIS  Equ     81             ;load char cnt reg w/char bufr Init Size
                = 103
002F            = 104 ENDBUF  EQU     2FH            ;END OF CHAR BUFFER
000B            = 105 ASBfSz  EQU     0BH            ;Accelerate stpr mtr buf count
000A            = 106 DSBfSz  Equ     0AH            ;Decelerate stpr mtr buf count
002F            = 107 SMBFST  EQU     2FH            ;STPR MTR BUFFER START
0025            = 108 SMBEnd  Equ     25H            ;Stpr Mtr Data Memory Address end
007F            = 109 DMTop   Equ     7FH            ;Data Memory Top
005D            = 110 DMSize  Equ     93             ;Data Memory Size (less two working reg's)
                = 111
0020            = 112 LastPh  Equ     20H            ;last phz psuedo reg addr
0021            = 113 CPSAdr  EQU     21H            ;CR phz psuedo reg
0022            = 114 LPSAdr  Equ     22H            ;LF phz psuedo reg
0023            = 115 PTAscS  Equ     23H            ;Char Print Test code start tmp store
                = 116
                = 117 ; PG
                = 118 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                = 119 ;      Register allocation
                = 120 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                = 121
                = 122 ;       All Indirect Data Memory Addressing via @Rn Inst must use
                = 123 ;         only registers 0 & 1 of either register bank.  Any other will
                = 124 ;         be rejected by the Assembler
                = 125 ;       Last character in lable indicates Register Bank referenced
                = 126
                = 127 ;      Register Bank 0
                = 128 ; ------------------------------------------------------------------------
0000            = 129 TmpR00  Equ     R0             ;RB0 Temporary Register
0001            = 130 TStrR0  EQU     R1             ;Store Time Register RB0
0002            = 131 GStR20  EQU     R2             ;General Status Register RB0
0003            = 132 PhzR30  EQU     R3             ;Stpr Mtr Phase Register RB0
0004            = 133 CntR40  Equ     R4             ;Count Reg. Phase count-Stpr Mtr loops
                = 134                                ;  Accel/Decel Count
0005            = 135 TConR0  Equ     R5             ;Time constant reg RB0
0006            = 136 LnCtR0  Equ     R6             ;Line count
                = 137
0007            = 138 OpnR70  EQU     R7             ;available
                = 139
                = 140 ;       Register Bank 0 Data Memory Address
                = 141 ; ------------------------------------------------------------------------
```

```
0000          TmpA00  Equ     00H             ;Temporary Register DM address
0001          TStrA0  EQU     01H             ;Time Store Register  DM address
0002          GStRAd  Equ     02H             ;RBO Char Status Reg  DM address
0003          PhzA20  EQU     03H             ;Stpr Mtr Phase Register DM address
0004          CntRA0  Equ     04H             ;Count Reg. Phase count-Stpr Mtr loops
                                              ;  Accel/Decel Count DM address
0005          TConA0  Equ     05H             ;Time constant reg DM address
0006          LnCtA0  Equ     06H             ;Line Count Register DM address

0007          OpnA70  EQU     07H             ;available


              ; PG

              ; ------------------------------------------------------------------
              ;       RBO  Status Byte Bit Definition
              ; ------------------------------------------------------------------

              ;       Bit             Definition
              ;       ----------------------------------------------------------
              ;       7               Stpr Mtr Direction:  L-to-R = 1, R-to-L = 0
              ;       6               1 = Sink / 0 = Not Sinked, Print Head Init and Fire
              ;       5               Stpr Mtr at speed and CR not left of Home
              ;       4               Accel/Decel Init, 1 = Done / 0 = Not Done

              ;       3               1 = FailSafe / 0 = Constant, Time Window
              ;       2               1 = Form Feed / 0 = Line Feed
              ;       1               1 = Do Not Print / 0 = Print
              ;       0               FAccel/DAccel drive Ready = 1/NotRdy = 0 (exit
              ;                         drive & decel stpr mtr)

              ;       Bit Masks:      RBO
              ;       Stepper Motor control bit masks function on GStR10
              ; ------------------------------------------------------------

              LRPrnt  Equ     80H             ;Left to Right Printing (ORL)
              RLPrnt  Equ     7FH             ;Right to Left Printing (ANL)
              SnkSet  Equ     40H             ;Ready Print flag
              ClrSnk  Equ     0BFH            ;clear Ready to Print Bit
              AtSpdF  Equ     20H             ;Stpr Mtr at constant speed
              NAtSpd  Equ     0DFH            ;Stpr Mtr Not at speed
              ADIntD  Equ     10H             ;Accel/Decel Init Done
              ADIntN  Equ     0EFH            ;Accel/Decel Init Not Done

              FsCTm   Equ     08H             ;FailSafe/Constant Time
              ClrFsC  Equ     0F7H            ;Clear FailSafe/Const time flag
              FrmFd   Equ     04H             ;do formfeed
              LineFd  Equ     0FBH            ;do line feed
              DoNotP  Equ     02H             ;set Do Not Print Stat bit
              OkPrnt  Equ     0FDH            ;Reset - Ok to Print
              Ready   Equ     01H             ;Ready drive Stpr Mtr
              NotRdy  Equ     0FEH            ;Not Ready exit Stpr Mtr drive


              ; PG

              ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
              ;       Register allocation   (cont)
              ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *


              ;       Register Bank 1
              ; ------------------------------------------------------------
0000          TmpR10  Equ     R0
0001          CAdrR1  EQU     R1              ;char data memory addr register
0002          ChStR1  EQU     R2              ;char processing status byte register
0003          CDtCR1  EQU     R3              ;Char Dot count register
0004          CDotR1  Equ     R4              ;Char dot temp storage register
0005          CCntR1  Equ     R5              ;Char count temp register
0006          StrCR1  EQU     R6              ;Store Char Register

0007          OpnR71  EQU     R7              ;Available

              ;       Register Bank 1 Data Memory Address
              ; ------------------------------------------------------------
```

```
0018          TmpA10  Equ     24              ;temporary/scratch register
0019          ChARR1  EQU     25              ;char data memory addr register
001A          ChStAd  Equ     26              ;RB1 Char Status Reg address
001B          CDtCA1  EQU     27              ;Char Dot count register
001C          CDotA1  Equ     28              ;Char dot temp storage register
001D          CCntA1  Equ     29              ;Char count temp register
001E          StrCA1  EQU     30              ;Store Char Register

001F          OpnA71  EQU     31              ;Available


              ; PG

              ; ---------------------------------------------------------------------
              ;       RB1  Status Byte Bit Definition
              ; ---------------------------------------------------------------------

              ;       Bit             Definition
              ;       -----           ----------------------------------------
              ;       7               Stpr Mtr Direction:  L-to-R = 1, R-to-L = 0
              ;       6               Char Init, 1 =  Done / 0 = Not Done
              ;       5               Char Lookup Table Page: 1 = Pg1, 0 = Pg2
              ;       4               1 = Test / 0 = Normal char print/input

              ;       3               1 = EOF / 0 = Not EOF
              ;       2               Full = 1/Not Full = 0, Line in Char Buffer
              ;       1               1 = CR/(LF) / 0 = Not CR/(LF)
              ;       0               1 = Init / 0 = Do Not Init, CB registers done

              ;       Bit Masks:      RB1
              ;       Character printing bit masks function on ChStR1
              ; ---------------------------------------------------------------
0080          ChrPrn  Equ     80H             ;Stpr Mtr Direction:  L-to-R = 1
007F          ClrCPr  Equ     7FH             ;Stpr Mtr Direction:  R-to-L = 0
0040          ChIntD  Equ     040H            ;Set Char Init Done
00BF          CIntND  Equ     0BFH            ;Reset Char Init Not Done
0020          ChOnP1  Equ     20H             ;Page 1 char, set rentry bit (ORL)
00DF          ChOnP2  Equ     0DFH            ;Page 2 char, reset rentry bit (ANL)
0010          TstPrn  Equ     10H             ;Char print test
00EF          NrmPrn  Equ     0EFH            ;Normal char input

0008          EOF     Equ     08H             ;set EOF Flag
00F7          ClrEOF  Equ     0F7H            ;clear EOF flag - Not EOF
0004          CRLF    Equ     04H             ;CR/LF
00FB          ClrCR   Equ     0FBH            ;Clear CR/LF
0002          CBFLn   Equ     02H             ;Full Line in Char Buffer
00FD          NCBFLn  Equ     0FDH            ;Not Full Line in Char Buffer
0001          IntCBR  Equ     01H             ;Init of CB registers done
00FE          ClICBR  Equ     0FEH            ;Init of CB registers not done


              ; PG

              ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
              ;               Equates  (cont)

              ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

              ;       Misc
              ; ---------------------------------------------------------------
0004          RLPShf  Equ     04H             ;R-to-L print lookup table addr shift

0020          Ascii   Equ     20H             ;hex nmbr of first Ascii Char
007F          AscLst  Equ     7FH             ;hex nmbr of last Ascii Char

00F3          CRCpl   Equ     0F3H            ;ASCII control code 2's complement
00F6          LFCpl   Equ     0F6H            ; "
00F4          FFCpl   Equ     0F4H            ; "
00E5          EscCpl  Equ     0E5H            ; "
00E0          AscCpl  Equ     0E0H            ; "
00C8          FTCpl   Equ     0C8H            ; "
000D          CR      Equ     0DH             ;Ascii code (hex)
0020          Space   Equ     20H             ;Ascii code (hex)

0081          LAsEnd  Equ     81H             ;Ascii End 2's cpl - test line start
0082          PAsEnd  Equ     82H             ;Ascii End 2's cpl - within line print
007F          AscStp  Equ     7fH             ;Ascii mask, strip off MSB
0042          PgLnCt  Equ     66              ;Page Line Count: Default = 66
00C4          PgLCpl  Equ     0C4H            ;Printed lines per page test
001B          EOFCpl  Equ     1BH             ;EOF ascii code cpl

              ;       Loop count values
              ; ---------------------------------------------------------------
```

```
0006          NDtCCt  Equ     06H             ;Normal Dot Column Count
000A          EDtCCt  Equ     0AH             ;Expanded Dot Column Count
0004          PHCnt1  EQU     04H             ;NUMBER OF SM PHASES ON INIT
0004          ILFCnt  Equ     04              ;Init LF step/phz count
0024          LPI6p6  Equ     36              ;Lines Per Inch 6.6
001B          LPI8p8  Equ     27              ;Lines Per Inch 8.8
0018          LPI10   Equ     24              ;Lines Per Inch 10
0001          LineCt  Equ     01              ;linefeed count
0042          FmFdCt  Equ     66              ;lines per formfeed count
0003          Status  EQU     03H             ;SEE BELOW FOR STATUS BYTE DEF.
                                              ;  TEST:  SET FOR CR STPR MTR CONTROL

                ; PG

                ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                ;       TIMER VALUES - UPI Timer/Counter is UP Counter
                ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                ;       12 MHz Clk timings
                ; ---------------------------------------------------------------
0080          DLYCL   EQU     80H             ;DELAY COUNT Long
0030          DLYCS   EQU     30H             ;DELAY COUNT Short
00CC          DlyTim  EQU     256-52          ;TIME DELAY constant ~2.0mS
0000          FailTm  EQU     256-256         ;FailSafe TIME = ~17.0mS
00CC          CrTmr1  EQU     256-52          ;CR Stpr Mtr Phase TIME = ~2.08mS
00BA          CrTmr2  EQU     256-70          ;CR Stpr Mtr Phase TIME = ~2.40mS
0092          CrTmr3  EQU     256-110         ;CR Stpr Mtr Phase TIME = ~4.16mS
00C0          IntTm2  EQU     256-64          ;Init Stpr Mtr Phase TIME = ~2.40mS
0098          LFTMR1  EQU     256-104         ;LF Stpr Mtr Phase TIME = ~4.16mS

                ;       I/O port bit masks
00DF          NotBsy  Equ     0DFH            ;Not Busy
0020          Busy    Equ     20H             ;Busy
00EF          Ack     Equ     0EFH            ;Ack
0010          ReSAck  Equ     10H             ;ReSet Ack

                ;       Misc bit Masks
000C          StrpLF  Equ     0CH             ;Strip off all bits but LF Stpr Mtr
0003          StrpCR  Equ     03H             ;Strip off all bits but CR Stpr Mtr

                ;       Print Head fires on low going edge of Trigger
                ;         bit #9 in dot column is masked off, always:  P2, bit 6
0040          PTRGLO  EQU     40H             ;PH TRIGGER BIT - LOW
00C0          PTRGHI  EQU     0C0H            ;PH TRIGGER BIT - HIGH

                ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                ;       Stepper Motor Phase State Equates
                ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

                ;       Stepper Motor Phase Shift Index Offset Offset
0000          FStCRP  EQU     00H             ;F CR stpr mtr phase data start addr
0003          RStCRP  EQU     03H             ;R CR stpr mtr phase data start addr
0008          STLFF   Equ     08H             ;Paper feed stpr mtr phase data start addr

                ;       CARRIAGE STEPPER MOTOR PHASE EQUATES
                ;       Forward (1 thru 4)  &  Reverse (4 thru 1) :
0001          CRMFP1  EQU     01B             ;CR STPR MTR PHASE 1
0003          CRMFP2  EQU     11B             ;CR STPR MTR PHASE 2
0002          CRMFP3  EQU     10B             ;CR STPR MTR PHASE 3
0000          CRMFP4  EQU     00B             ;CR STPR MTR PHASE 4

                        ;LINE FEED STEPPER MOTOR PHASE EQUATES
                ;       Forward:
0004          LFMFP1  EQU     0100B           ;LF STPR MTR PHASE 1
000C          LFMFP2  EQU     1100B           ;LF STPR MTR PHASE 2
0008          LFMFP3  EQU     1000B           ;LF STPR MTR PHASE 3
0000          LFMFP4  EQU     0000B           ;LF STPR MTR PHASE 4

                ; PG
```

```
                = 365 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                = 366 ;         STEPPER MOTOR SELECT & CONTROL [CURRENT LIMITING]
                = 367 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                = 368
                = 369 ;         PORT BIT ASSIGNMENT:
                = 370
                = 371 ;                 \ \ \
                = 372 ;                 S S S -
                = 373 ;                 L C C
                = 374 ;                 F R R
                = 375 ;                   8 1
                = 376 ;                   0 3
                = 377 ;                     2
                = 378 ;         ------------------
                = 379 ;                 5 5 5 5
                = 380 ;                 3 2 1 0
                = 381 ;         ------------------
                = 382 ; CODING:
                = 383 ;         SLF     0 1 1 0         06H
                = 384 ;         SCR80   1 0 0 0         0AH
                = 385 ;         SCR132  1 1 0 0         0CH
                = 386 ;         SMOFF   1 1 1 0         0EH
                = 387 ;                 W/SCR80 & SCR132 '0' [BOTH SELECTED]
                = 388 ;                   DEFAULT IS TO 80 COL.
                = 389 ;                   [DO NOT KNOW WHETHER SCR80='0' WILL
                = 390 ;                   SELECT 80 COL ONLY] - REQUIRES TEST.
                = 391
0008            = 392 SCR80   EQU     08H             ;SELECT CR STPR MTR - 80 COL
                = 393                                 ;  w/LF STPR MTR OFF
000C            = 394 SCR132  EQU     0CH             ;SELECT CR STPR MTR - 132 COL
                = 395                                 ;  w/LF STPR MTR OFF
0006            = 396 SLF     EQU     06H             ;SELECT LF STPR MTR ON
                = 397                                 ;  w/CR STPR MTR OFF
000E            = 398 SMOFF   EQU     0EH             ;SELECT CR & LF STPR MTR OFF
                  399
                  400
                  401 ; PG
                  402 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                  403 ;                 MAIN PROGRAM BODY
                  404 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                  405
                  406 ;         Power On / Reset Program Entry
                  407
                  408 ;         PROGRAM START
                  409
0000              410           Org     00H
                  411
0000 040B         412 START:    JMP     RESET
                  413
                  414 ;         INPUT BUFFER FULL INTERRUPT CALL ENTRY AND VECTOR
0003              415           ORG     03H
0003 1425         416 IBFIV:    Call    IBFIS
0005 93           417           RETR
                  418 ;         TIMER OVERFLOW INTERRUPT CALL ENTRY AND VECTOR
0007              419           ORG     07H
0007 1429         420 TMRIV:    Call    TMRIS
0009 C5           421           SEL     RB0
000A 83           422           Ret
                  423
                  424 ;                 INITIALIZATION
                  425
000B 15           426 ReseT:    Dis     I
000C 35           427           Dis     TCntI
000D B40F         428           Call    InitAl          ;set all critical outputs inactive
000F B42F         429           Call    ClrDM           ;clear all data memory - 93H to 7FH
                  430                                   ;  do not clear RB0, RB1 or Stack
0011 B44B         431           Call    InitCR          ;CALL CR SM POWER ON INIT
0013 9400         432           Call    InitLF          ;CALL LF SM POWER ON INIT
                  433
                  434 ;                 MAIN PROGRAM LOOP
                  435 ;                 All program segments are called from here
                  436
0015 B422         437 Home:     Call    CRHome          ;Call Home CR routine -
                  438                                   ;  fixes logical and physical CR Home
0017 B400         439           Call    Defalt          ;set default register values
0019 142C         440 CBInpt:   Call    ESCBff          ;Stat Switch / CB Input Service Test
                  441                                   ; test for:  CB full/fill, LF, FF,
                  442                                   ;            Char Prnt Test
```

```
001B 3400   Repeat: Call    SMDriv          ;Call Forward Stpr Mtr Drive
001D 940D           Call    LFDriv          ;Call Linefeed Stpr Mtr Drive
001F D5             SEL     RB1
0020 FA             Mov     A,ChStR1        ;get the Char Status Register RB1
0021 7215           JB3     Home            ;jump to CR SM Home if EOF bit set
0023 0419           Jmp     CBInpt          ;loop to Char Buffer Input test

; PG
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;                   Interrupt Service Routine
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

; ---------------------------------------------------------------------------
;                   Input Buffer Full Interrupt Service Routine
; ---------------------------------------------------------------------------

IBFIS:
; ---------------------------------------------------------------------------
;           Acknowledge Char input and set Hold/Busy Active
0025 8A20           ORL     P2,#Busy        ;get & set DBB ACK/Busy Bits
0027 15             Dis     I               ;disable IBF interrupts
0028 83             Ret

; ---------------------------------------------------------------------------
;                   Timer / Counter Interrupt Service Routine
; ---------------------------------------------------------------------------
;           ITF interrupt service routine disables all intr during
;             stpr mtr phase shifting
0029 15     TMRIS:  Dis     I               ;disable IBF interrupts
002A 35             Dis     TCntI           ;dicable ITF interrupts
002B 83             Ret

; PG
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;           External Status Switch Check/Char. Buffer Fill
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
ESCBfF:                                     ;Prep for normal character handling/input
002C D5             SEL     RB1
002D FA             Mov     A,ChStR1        ;get the character stat reg byte
002E 53EF           ANL     A,#NrmPrn       ;set normal character input
0030 AA             Mov     ChStR1,A        ;store the stat byte
0031 C5             SEL     RB0

;           Test External Status Port
0032 0F             MovD    A,P7            ;get the stat switch port bits
0033 123D           JB0     FormFd          ;  service Formfeed
0035 3245           JB1     LinFd           ;  service Linefeed
0037 5249           JB2     ChrTst          ;  service Character TEST
0039 725E           JB3     OnLine          ;  service Char Buffer Check/Fill
003B 042C           Jmp     ESCBfF          ;Loop
; ---------------------------------------------------------------------------
003D FA     FormFd: Mov     A,GStR20        ;get the status byte
003E 4304           ORL     A,#FrmFd        ;set the formfeed stat flag
0040 AA             Mov     GStR20,A        ;store trhe status byte
0041 940D           Call    LfDriv          ;do a formfeed
0043 042C           Jmp     ESCBfF
; ---------------------------------------------------------------------------
0045 940D   LinFd:  Call    LfDriv          ;do a line drive
0047 042C           Jmp     ESCBfF
; ---------------------------------------------------------------------------
0049 D5     ChrTst: SEL     RB1
004A FA             Mov     A,ChStR1        ;get the character stat reg byte
004B 4310           ORL     A,#TstPrn       ;set character test flag
004D AA             Mov     ChStR1,A        ;store the stat byte
004E B823           Mov     TmpR10,#PTAscS  ;load the psuedo Ascii code tmp reg addr
0050 F0             Mov     A,@TmpR10       ;get the inc'd ascii code
0051 0381           ADD     A,#LAsEnd       ;test for code end
0053 9657           JNZ     AscCLd          ;if not code end jmp to load
                                            ;if end restart ascii at begining
0055 B020           Mov     @TmpR10,#Ascii  ;store the ascii code start
0057 F0     AscCLd: Mov     A,@TmpR10       ;get the ascii code again
0058 AF             Mov     OpnR7i,A        ;place in the empty register
0059 10             Inc     @TmpR10         ;Inc start ASCII char in data memory
005A B439           Call    PrnTst          ;call the DM load procedure
005C C5             SEL     RB0             ;reselect reg bank 0
005D 83             Ret
; ---------------------------------------------------------------------------
```

```
005E D5          OnLine: SEL     RB1             ;select char buffer registers
005F 05                  EN      I               ;enable interrupts
0060 FA          CBfCk1: Mov     A,ChStR1        ;get the Char Stat Byte
0061 3267                JB1     CBCkEx          ;if Chr Buf has full line exit
0063 146D        IBfCk:  Call    CBFill          ;read a char into Char Buffer
0065 0460                Jmp     CBFCk1          ;loop to Char Buf Ful test
0067 C5          CBCkEx: SEL     RB0
0068 83                  Ret

                 ; PG
                 ; ---------------------------------------------------------------------
                 ;       Character Input
                 ; ---------------------------------------------------------------------
                 ;       Input Buffer Full service routine:  test for Char buffer full-exit
                 ;         else load char into char buffer
0069 D5          IBFSrv: SEL     RB1
006A FA                  Mov     A,ChStR1        ;get the RB0 stat byte
006B 32EC                JB1     CBFull          ;if Do Not Print Bit Set - EXIT
006D 527C        CBFill: JB2     CBPad           ;test for CB padding flag
                                                 ;  if not pad enable char input
                                                 ;  tell the host to send char's
006F 05                  EN      I
0070 D6EC                JNIBF   CBF1Ex
                 ; ---------------------------------------------------------------------
                 ;       Acknowledge Char input and set Hold/Busy Active
0072 FA                  Mov     A,ChStR1        ;get the RB1 Char Stat Byte
0073 127C                JB0     SkpInt          ;test for CB has been Initialized
                 ; ---------------------------------------------------------------------
                 ;       Init of all Char handling registers
0075 4301                ORL     A,#IntCBR       ;set CB Reg skip Initialization stat bit
0077 AA                  Mov     ChStR1,A        ;save the altered stat byte
0078 B97F                Mov     CAdrR1,#FCBfSt  ;load char reg w/char bufr strt
007A BD50                Mov     CCntR1,#ChBfSz  ;load char cnt reg w/char bufr size
                 CBPad:
007C ED86        SkpInt: DJNZ    CCntR1,LdChar   ;DECREMENT BUFFER SIZE
007E FA                  Mov     A,ChStR1        ;get the status byte
007F 4302                ORL     A,#CBFLn        ;set Char Buffer Full Line stat bit
0081 53FB                ANL     A,#ClrCr        ;clear the CR/(LF) stat bit
0083 53FE                ANL     A,#ClICBR       ;reset CB Init bit: init CB reg on entry
0085 AA                  Mov     ChStR1,A        ;store the status byte
0086 FA          LdChar: Mov     A,ChStR1        ;get the status byte
0087 52E1                JB2     CBPad1          ;CB not full but CR/LF previously
                                                 ;  received so pad CB
0089 9AEF                ANL     P2,#Ack         ;output DBB Ack low
008B 22                  In      A,DBB           ;read the Char
008C 537F                ANL     A,#AscStp       ;strip off MSB
008E A8                  Mov     TmpR10,A        ;temp save char
008F 8A10                ORL     P2,#ReSAck      ;output DBB ACK High
                 ; ---------------------------------------------------------------------
                 ;       test for ASCII printable character
0091 03E0                ADD     A,#ASCCpl       ;test for Carriage Return
0093 F697                JC      AsciiC          ;jmp to service
0095 049C                Jmp     ChrChk
0097 97          AsciiC: Clr     C               ;clear carry flag
0098 F8                  Mov     A,TmpR10        ;get the char back
0099 A1                  Mov     @CAdrR1,A       ;load data memory w/Char
009A 04E3                Jmp     IBFSrE
                 ; ---------------------------------------------------------------------
                 ;       test for CR/LF:  if CR/LF Strip off LF and exit setting
                 ;         Char Buffer Init Stat bit
009C F8          ChrChk: Mov     A,TmpR10        ;get the char back
009D 03F3                ADD     A,#CRCpl        ;test for Carriage Return
009F C6C3                JZ      CRChr           ;  if CR go service it
00A1 F8                  Mov     A,TmpR10        ;get the char back
00A2 031B                ADD     A,#EOFCpl       ;test for End Of File
00A4 96AA                JNZ     ChrCk1          ;if not EOF jmp to CB Pad
00A6 F8                  Mov     A,TmpR10        ;if EOF, place it in CB
00A7 A1                  Mov     @CAdrR1,A       ;load data memory w/CR Char
00A8 04B9                Jmp     ExtSet          ;Exit
00AA F8          ChrCk1: Mov     A,TmpR10        ;get the status byte
00AB 03F4                ADD     A,#FFCpl        ;test for FormFeed
00AD 96E1                JNZ     CBPad1          ;if not FF Pad the CB
00AF C5                  SEL     RB0
00B0 FA                  Mov     A,GStR20        ;get the status byte
00B1 4304                ORL     A,#FrmFd        ;set the formfeed flag
00B3 AA                  Mov     GStR20,A        ;store the status byte
00B4 D5                  SEL     RB1
00B5 FA                  Mov     A,ChStR1        ;get the status byte
00B6 4304                ORL     A,#CRLF         ;set CRLF stat bit:  pad balance of CB
                                                 ;  with Spaces until fill
00B8 AA                  Mov     ChStR1,A        ;store the status byte
00B9 FA          ExtSet: Mov     A,ChStR1        ;get the status byte
```

```
00BA 4302          ORL     A,#CBFLn        ;set Char Buffer Full Line stat bit
00BC 53FB          ANL     A,#ClrCr        ;clear the CR/(LF) stat bit
00BE 53FE          ANL     A,#ClICBR       ;reset CB Init bit: init CB reg on entry
00C0 AA            Mov     ChStR1,A        ;store the status byte
00C1 04EC          Jmp     CBFlEx          ;Exit
                   ; ---------------------------------------------------------------------
                   ;       Store CR char read in LF char (assume its always there) and ignor it
00C3 F8            CRChr:  Mov     A,TmpR10        ;get the char back
00C4 A1                    Mov     @CAdrR1,A       ;load data memory w/CR Char
00C5 C5                    SEL     RB0
00C6 1E                    INC     LnCtR0          ;inc the line count
00C7 FE                    Mov     A,LnCtR0        ;get the line count
00C8 03C4                  Add     A,#PgLCpl       ;test for page feed ln cnt
                                                   ;  if LnCt => PgLnCt set formfeed flag
00CA E6D0                  JNC     NoFmFd          ;if not at end of page skip
00CC FA                    Mov     A,GStR20        ;get the status byte
00CD 4304                  ORL     A,#FrmFd        ;set the form feed status flag
00CF AA                    Mov     GStR20,A        ;save the status byte
00D0 D5            NoFmFd: SEL     RB1
00D1 05                    En      I               ;enable the IBF service
00D2 9ADF                  ANL     P2,#NotBsy      ;output a not busy to Host
00D4 D6D4          LFTest: JNIBF   LFTest          ;loop to next char
00D6 9AEF                  ANL     P2,#Ack         ;output DBB Ack low
00D8 22                    In      A,DBB           ;get next Char - assume it's a LF
                                                   ;  and ignor it (LF is forced upon
                                                   ;  detection of CR at print time).
00D9 FA            SetPad: Mov     A,ChStR1        ;get the status byte
00DA 4304                  ORL     A,#CRLF         ;set CRLF stat bit:  pad balance of CB
                                                   ;  with Spaces until fill
00DC AA                    Mov     ChStR1,A        ;store the status byte
00DD 8A10                  ORL     P2,#ReSAck      ;output DBB ACK High
00DF 04E3                  Jmp     IBfSrE          ;jmp to addr step & exit
                   ; ---------------------------------------------------------------------
                   ;       fill Char Buffer with space
00E1 B120          CBPad1: Mov     @CAdrR1,#Space  ;load data memory w/Char
                   ; ---------------------------------------------------------------------
                   ;       step the char address test for CB full &/or pad
00E3 C9            IBFSrE: DEC     CAdrR1          ;Decrement dat memory location
00E4 FA                    Mov     A,ChStR1        ;get the status byte
00E5 32EC                  JB1     CBFull          ;test for CB Full
00E7 52EC                  JB2     CBFlEx          ;test for CB pad - exit w/Busy set
                   ; ---------------------------------------------------------------------
                   ;       Set Busy Line Low - Not Busy
00E9 05                    EN      I
00EA 9ADF                  ANL     P2,#NotBsy      ;output a not busy to Host
                   ; ---------------------------------------------------------------------
                   ;       exit w/ Busy Still set high
                   CBFull:
00EC 83            CBFlEx: Ret

                   ; PG
                   ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                   ;       L-to-R/R-to-L Carriage Stepper Motor Drive
                   ;          and Line Printing
                   ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

0100                       ORG     100H

0100 3622          SMDriv: JTO     RAccel          ;if Print Head at left drive right
                                                   ;  else drive left
                   ; F=================================================================
                   FAccel:                         ;L-to-R Accelerate Stepper Motor
                   ;       Set the Forward acceleration/drive Entry status bits
0102 FA                    Mov     A,GStR20        ;get the status byte
0103 53BF                  ANL     A,#ClrSnk       ;set not at speed flag = 0
0105 53DF                  ANL     A,#NAtSpd       ;set Not At Speed flag = 0
0107 4380                  ORL     A,#LRPrnt       ;set L-to-R prnt stat bit = 1
0109 4301                  ORL     A,#Ready        ;set stpr mtr ready - Drive On
010B 53EF                  ANL     A,#ADIntN       ;set A/D Init Not Done
010D AA                    Mov     GStR20,A        ;store the status byte
010E D5            CBRDir: SEL     RB1
010F FA                    Mov     A,ChStR1        ;get the Char Stat Reg Data Mem Addr
0110 4380                  ORL     A,#LRPrnt       ;Set L-to-R print bit
0112 AA                    Mov     ChStR1,A        ;save the Char Stat byte
0113 C5                    SEL     RB0

                           Restore the phase register index addresses
0114 B821                  Mov     TmpR00,#CPSAdr  ;get Phz Storage Addr psuedo reg
0116 F0                    Mov     A,@TmpR00       ;  get stored CR last phase index addr
0117 AB                    Mov     PhzR30,A        ;place last LF phase index addr in Phz Reg
```

```
                ;       Set up for next phase bit output before entering timing loops
0118 1B                 INC     PhzR30          ;STEP PHASE DB ADDRESS
0119 FB                 MOV     A,PhzR30        ;CHECK THE PHASE COUNT REG
011A 521E               JB2     IAFZrP          ;CHK FOR COUNT BIT ROLLOVER
011C 2440               JMP     SMDflt          ;skip adr index reset
011E BB00       IAFZrP: MOV     PhzR30,#FStCRP  ;ZERO CR SM PHASE REGISTER
0120 2440               Jmp     SMDflt

                ; R=================================================================
                RAccel:         ;R-to-L Accelerate Stepper Motor
                ; ---------------------------------------------------------------
                ;       Set the Reverse acceleration/drive Entry status bits
0122 FA                 Mov     A,GStR20        ;get the status byte
0123 53BF               ANL     A,#ClrSnk        ;clear Print Ready bit
0125 53DF               ANL     A,#NAtSpd       ;set Not At Speed flag = 0
0127 537F               ANL     A,#RLPrnt       ;set R-to-L prnt status bit
0129 4301               ORL     A,#Ready        ;set stpr mtr ready - Drive On
012B 53EF               ANL     A,#ADIntN       ;set A/D Init Not Done
012D AA                 Mov     GStR20,A        ;store the status byte
012E D5         RCBRDr: SEL     RB1
012F FA                 Mov     A,ChStR1        ;get the Char Stat Reg Data Mem Addr
0130 537F               ANL     A,#RLPrnt       ;Set R-to-L print bit
0132 AA                 Mov     ChStR1,A        ;save the Char Stat byte
0133 C5                 SEL     RB0
                ; ---------------------------------------------------------------
                ;       Restore the  phase register index address
0134 B821               Mov     TmpR00,#CPSAdr   ;get Phz Storage Addr psuedo reg
0136 F0                 Mov     A,@TmpR00       ;  get stored CR last phase index addr
0137 AB                 Mov     PhzR30,A        ;place last LF phase index addr in Phz Reg
                ;       Set up for next phase bit output before entering timing loops
0138 CB                 Dec     PhzR30          ;STEP PHASE DB ADDRESS
0139 FB                 MOV     A,PhzR30        ;CHECK THE PHASE COUNT REG
013A 523E               JB2     IARZrP          ;CHK FOR COUNT BIT ROLLOVER
013C 2440               JMP     SMDflt
013E BB03       IARZrP: MOV     PhzR30,#RStCRP  ;ZERO CR SM PHASE REGISTER

                SMDflt:
                ; ---------------------------------------------------------------
                ;       for stablization of unused stpr mtr during CR stpr mtr drive,
                ;         store the unused stpr mtr current phase bits
0140 B822               Mov     TmpR00,#LPSAdr   ;get the CR phz storeage addr
0142 F0                 Mov     A,@TmpR00       ;get the byute stored there
0143 E3                 MovP3   A,@A            ;get the phz data byte
0144 B820               Mov     TmpR00,#LastPh  ;load Last Phz psuedo reg to Temp Reg
0146 A0                 Mov     @TmpR00,A       ;store Last Phase bits - inderect

                ;       SetUp Stpr Mtr Time Constant
0147 BDBA               MOV     TConR0,#CrTmr2  ;Load time constant Reg

                Select:                         ;Select the Stpr Mtr
0149 2308               MOV     A,#SCR80        ;GET CR SM SELECT BITS
014B 3D                 MOVD    P5,A            ;SELECT SM [SCR80]
                ; ---------------------------------------------------------------
                ;       SetUp Stpr Mtr Phase Shift index address register
                ;       Output next phase and init timer to Std Time constant
014C FD         STRTT:  MOV     A,TConR0        ;get time constant from reg
014D 62                 MOV     T,A             ;load the timer
014E FB                 MOV     A,PhzR30        ;get the phz reg indirect addr index
014F E3                 MovP3   A,@A            ;do indirect get of phz bits

                ; ---------------------------------------------------------------
                ;       patch together the CR last and LF next phase bits
0150 B820               Mov     TmpR00,#LastPh  ;load Last Phz psuedo reg to Temp Reg
0152 40                 ORL     A,@TmpR00        ;patch together CR existing & new LF
0153 3C                 MOVD    P4,A            ;OUTPUT BITS
0154 55                 STRT    T               ;START TIMER

                ;       At start of timing loop do all Stpr Mtr Accel/Decel or
                ;         Character SetUp overhead
0155 740C               Call    ADPTst          ;call Accel/Decel/Print Test

                ;       Set up for next phase bit output before entering timing loops
                PNRdy1: ;test for forward / reverse phase start indirect index to load
0157 FA                 Mov     A,GStR20        ;store stat byte
0158 F264               JB7     AclF2

                ; reverse:
                ;       Set up for next phase bit output before entering timing loops
015A CB                 Dec     PhzR30          ;STEP PHASE DB ADDRESS
015B FB                 MOV     A,PhzR30        ;CHECK THE PHASE COUNT REG
015C 5260               JB2     ARZroP          ;CHK FOR COUNT BIT ROLLOVER
015E 2462               JMP     ARNxtP
0160 BB03       ARZroP: MOV     PhzR30,#RStCRP  ;ZERO CR SM PHASE REGISTER
0162 246C       ARNxtP: Jmp     ANxtPh
```

```
                 767
                 768 ; forward:
                 769 ;        Set up for next phase bit output before entering timing loops
0164 1B          770 AclF2:   INC    PhzR30          ;STEP PHASE DB ADDRESS
0165 FB          771          MOV    A,PhzR30        ;CHECK THE PHASE COUNT REG
0166 526A        772          JB2    AFZroP          ;CHK FOR COUNT BIT ROLLOVER
0168 246C        773          JMP    ANxtPh          ;skip adr index reset
016A BB00        774 AFZroP:  MOV    PhzR30,#FStCRP  ;ZERO CR SM PHASE REGISTER
                 775 ANxtPh:
                 776 ; ---------------------------------------------------------------
                 777 ;        stage one timer loop - T occurs before Std timeout
                 778 ;          wait for time out
016C 1682        779 TLOOP2:  JTF    FAILSF          ;JMP ON TIME OUT-t DOES NOT OCCUR 1ST
016E 5672        780          JT1    tCHK1           ;IS T HIGH-JMP TO tCHK
0170 246C        781          JMP    TLOOP2          ;LOOP FOR JT1 OR JTF
0172 00          782 tCHK1:   NOP                    ;delay, then double check T signal
0173 5677        783          JT1    tTruW1          ;JUMP T TEST TRUE-WAIT FOR JTF
0175 246C        784          JMP    TLOOP2
                 785 tTruW1:
                 786 ;        test for Print Ready bit - was Print Head Fire Setup Done?
                 787 ;        insert acceleration time/store time count done/notdone flag bit
0177 FA          788          Mov    A,QStR20        ;get the status byte - prep for prnt
0178 D27C        789          JB6    RdyPr2          ;if Ready Print bit set call PHFire
017A 247E        790          Jmp    SkpPHF          ;  else skip Print Head Fire
017C 74CA        791 RdyPr2:  Call   PHFire          ;print head solenoid fire routine
                 792 PNRdy2:
                 793 SkpPHF:
017E 1698        794 tTruW2:  JTF    NXTPHZ          ;JUMP TO SM ERROR
0180 247E        795          JMP    tTruW2          ; LOOP TO TLOOP3
                 796 ; ---------------------------------------------------------------
                 797 ;        Step into failsafe/startup timer setup - T does not
                 798 ;          occurs before Std Time timeout, load failsafe SM protection
                 799 ;          time and wait for failsafe timeout or T.  If T occurs
                 800 ;          output phase immediately after T verify.
0182 2300        801 FAILSF:  MOV    A,#FailTm       ;LOAD TIMER W/~15.0mS
0184 62          802          MOV    T,A             ;  SM PROTECTION TIMEOUT
0185 55          803          STRT   T               ;START TIMER
                 804 ; ---------------------------------------------------------------
                 805 ;        set the Status bit for Store time test
0186 FA          806          Mov    A,QStR20        ;get the status byte
0187 4308        807          ORL    A,#FSCTm        ;set Failsafe/constant time flag
0189 AA          808          Mov    QStR20,A        ;store the status byte
018A 5690        809 TLOOP3:  JT1    tCHK2           ;IS T HIGH
018C 16AC        810          JTF    DSLECT          ;IF TIME OUT GO SM ERROR
018E 248A        811          JMP    TLOOP3          ;LOOP UNTIL T HIGH OR T-OUT
0190 00          812 tCHK2:   NOP                    ;WAIT
0191 5695        813          JT1    StrTm1          ;jump out and store elapsed time
0193 248A        814          JMP    TLOOP3          ;  JMP TO FAILSF LOOP
0195 65          815 StrTm1:  Stop   TCnt            ;stop the failSafe Timer
0196 42          816          Mov    A,T             ;read the timer
0197 A1          817          MOV    @TStrR0,A       ;Store the time read in indexed addr
                 818                                 ; - next entry to A/D Memorize Time
                 819                                 ; routine will add time constant to it
                 820
                 821 ;     Test is CR Stpr Mtr Drive is finished prior to next phase output
                 822 ; ---------------------------------------------------------------
                 823 NXTPHZ:
                 824 ;        test for forward / reverse phase start indirect index to load
0198 FA          825          Mov    A,QStR20        ;store stat byte
0199 F2A7        826          JB7    FDrive
                 827 ; Reverse  --  test for Reverse Stpr Mtr Drive procedure exit
                 828 ;        ALWAYS drive the CR to the left most HOME position
019B 26AC        829          JNTO   EOLn            ;test if home position jmp stop
019D FA          830          Mov    A,QStR20        ;get the status byte
019E 124C        831          JB0    StrtT           ;test Ready stat bit:
                 832                                 ;  if bit 0 = 1 then Print More
01A0 4302        833          ORL    A,#DoNotP       ;set the do not print flag
01A2 53BF        834          ANL    A,#ClrSnk       ;clear Print Ready bit
01A4 AA          835          Mov    QStR20,A        ;save the status byte
01A5 244C        836          Jmp    StrtT           ;continue CR SM drive
                 837                                 ;  - only exit is HR
                 838 ; Forward  --  test for Forward Stpr Mtr Drive procedure exit
                 839 FDrive:
01A7 FA          840          Mov    A,QStR20        ;get the status byte
01A8 124C        841          JB0    StrtT           ;test Ready stat bit:
                 842                                 ;  if bit 0 = 1 then Print More
01AA 24AC        843          Jmp    EOLn            ;  else jmp to End Of Line exit
                 844                                 ;jump to start timer again
                 845 DSLECT:
01AC 5437        846 EOLn:    Call   DeclSM          ;call Sptr Mtr Deceleration
                 847 ; ---------------------------------------------------------------
                 848 ;        test for forward / reverse phase start indirect index to load
01AE FA          849          Mov    A,QStR20        ;store stat byte
01AF F2B3        850          JB7    FDrvFS          ;jmp to f drive flag set
```

```
01B1 53FD          ANL     A,#OkPrnt        ;reset print flag - Ok Print
                                            ; only if printing R-to-L

                   ;       update the status byte
01B3 53BF          FDrvFS: ANL     A,#ClrSnk        ;clear Print Ready bit
                                            ;set the Status bit for Store time test
01B5 53DF                  ANL     A,#NAtSpd        ;Clear At Print Speed Bit
01B7 AA                    Mov     QStR20,A         ;save the status byte
01B8 83                    RET

                   ; PG
                   ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                   ;       Stepper Motor Accel. Time Storeage
                   ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

0200                       ORG     200H
                                            ;  Entry has Gen Stat Byte in A
0200 920C          ADMmTS: JB4     DADInt           ;is A/D init done - then jmp

                   ;       1st Entry initializes the A/D Time store working registers
0202 B92F                  Mov     TStrR0,#SMBfSt   ;Load the Stpr Mtr Buffer Start Addr
0204 BC0B                  Mov     CntR40,#ASBfSz   ;Load the Buffer Size
0206 FA                    Mov     A,QStR20         ;get the status byte
0207 4310                  ORL     A,#ADIntD        ;set not 1st Accel Entry Flag
0209 AA                    Mov     QStR20,A         ;store the status byte
020A 4436                  Jmp     ADExit           ;exit - 1st entry has not generated
                                            ;  a closed time window

                   ;       Step the A/D Store count
020C EC26          DADInt: DJNZ    CntR40,StorCt    ;dec Times to store count
                                            ;if not 0 store the count
                                            ;else at end-set done flag
020E FA                    Mov     A,QStR20         ;get the status byte
020F 4320                  ORL     A,#AtSpdF        ;set at speed/no more to store flag
0211 AA                    Mov     QStR20,A         ;store the status byte

                   ;       Initialize Char Print Registers:  if printing enabled
0212 3226                  JB1     StorCt           ;if Do Not Print stat bit set
                                            ;  Skip the Char register init

                   ;       Initialize all Char Reg's
                   ;       Test for L-to-R (forward) or R-to-L (reverse) printing
0214 D5                    SEL     RB1
0215 FA                    Mov     A,ChStR1         ;get the status byte
0216 4340                  ORL     A,#CHIntD        ;set Char Init Done flag - bypass
0218 AA                    Mov     ChStR1,A         ;save the status byte
0219 F21F                  JB7     LdCBR1           ;test Chr Stat Byte Returned
                                            ;  if bit 7 = 1 then Print L-to-R
021B B92F          LdCBR:  Mov     CAdrR1,#RCBfIS   ;load char reg w/char bufr strt R-to-L
021D 4421                  Jmp     LdCBR2

021F B980          LdCBR1: Mov     CAdrR1,#FCBfIS   ;load char reg w/char bufr strt L-to-R

0221 BD51          LdCBR2: Mov     CCntR1,#ChBfIS   ;load char cnt reg w/char bufr size
0223 BB01                  Mov     CDtCR1,#01       ;set the chr dot column cnt
0225 C5                    SEL     RB0

                   ;       Test for t > Tc or t < Tc
0226 722C          StorCt: JB3     FailST           ;test for failsafe time switch

                   ;       t < Tc = store Time Constant in use
0228 FD                    Mov     A,TConR0         ;Get time constant currently in use
0229 A1                    Mov     @TStrR0,A        ;Memorize/Store the time - indirect addr
022A 4435                  Jmp     ADPRet

                   ;       t > Tc = store Time Constant + FailSafe Time Elapsed
                   ;       [see Accel/Cnst Speed/Decel WaveForm]
                   ;       equation is:     Trd - FailSafe Time = Tx
                   ;                     => Trd + Cpl(FailSafe Time) = Tx
                   ;                        Tx + Tcnst = T
                   ;                        Store/Memorize T

022C F1            FailST: Mov     A,@TStrR0        ;get the stored time
022D 03C8                  Add     A,#FTCpl         ;2's cpl add
022F 6D                    Add     A,TConR0         ;Add: Time stored + Time constant
                                            ;  currently in use
0230 A1                    Mov     @TStrR0,A        ;Memorize/Store the time
                   ;       Reset the Status bit for Store time test

0231 FA                    Mov     A,QStR20         ;get the status byte
0232 53F7                  ANL     A,#ClrFSC        ;reset Failsafe/constant time flag
                                            ;  assumes entry via constant time
```

```
0234 AA                 Mov     GStR20,A        ;store the status byte
0235 C9         ADPRet: Dec     TStrR0          ;step the A/D time data store addr
0236 83         ADExit: Ret

                ; PG

                ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                ;       Carriage Stepper Motor Deceleration

                ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

                DeclSM:
                ;       SetUp the Deceleration registers
0237 B925               Mov     TStrR0,#SMBEnd  ;Load the Stpr Mtr Buffer End Addr
0239 BC0A               Mov     CntR40,#DSBfSz  ;Load the Buffer Size
023B FB                 MOV     A,PhzR30        ;get phase index address
023C E3                 MovP3   A,@A            ;get phase from indexed address
                ;       patch together the CR last and LF next phase bits
023D B820               Mov     TmpR00,#LastPh  ;load Last Phz psuedo reg to Temp Reg
023F 40                 ORL     A,@TmpR00       ;patch together CR existing & new LF
0240 3C                 MOVD    P4,A            ;OUTPUT BITS
0241 F1         StrtTD: MOV     A,@TStrR0       ;get time from indexed data memory
0242 62                 MOV     T,A             ;load timer
0243 55                 STRT    T               ;START TIMER
0244 19                 Inc     TStrR0          ;step the Memorized time addr index reg
                ;       test for forward / reverse phase start indirect index to load
0245 FA                 Mov     A,GStR20        ;store stat byte
0246 F252               JB7     DclF2

                ; reverse:
                ;       Set up for next phase bit output before entering timing loops
0248 CB                 Dec     PhzR30          ;decrement the phase addr
0249 FB                 MOV     A,PhzR30        ;Get the phz data addr
024A 524E               JB2     DRZroP          ;CHK FOR COUNT BIT ROLLOVER
024C 445A               JMP     DNxtPh
024E BB03       DRZroP: MOV     PhzR30,#RStCRP  ;ZERO CR SM PHASE REGISTER
0250 445A               Jmp     DclR2

                ; forward:
                ;       Set up for next phase bit output before entering timing loops
0252 1B         DclF2:  Inc     PhzR30          ;increment the phase addr
0253 FB                 MOV     A,PhzR30        ;Get the phz data addr
0254 5258               JB2     DZroPh          ;CHK FOR COUNT BIT ROLLOVER
0256 445A               JMP     DNxtPh          ;skip adr index reset
0258 BB00       DZroPh: MOV     PhzR30,#FStCRP  ;ZERO CR SM PHASE REGISTER
                DNxtPh:                         ;set up for next phase shift
025A FB         DclR2:  MOV     A,PhzR30        ;get phase index address
025B E3                 MovP3   A,@A            ;get phase from indexed address
                ;       patch together the CR last and LF next phase bits
025C B820               Mov     TmpR00,#LastPh  ;load Last Phz psuedo reg to Temp Reg
025E 40                 ORL     A,@TmpR00       ;patch together CR existing & new LF
025F 1663       TLoopD: JTF     NxtPD2          ;JMP ON TIME OUT TO NEXT PH
0261 445F               JMP     TLoopD          ;LOOP UNTIL TIME OUT
0263 3C         NxtPD2: MOVD    P4,A            ;OUTPUT BITS
0264 EC41               DJNZ    CntR40,StrtTD   ;Exit Test

                ;       Set Storeage of next phase data in psuedo addr. This insures
                ;       next phase is sequence correct for stpr mtr drive direction
0266 B821       SetRN:  Mov     TmpR00,#CPSAdr  ;get Phz Storeage Addr psuedo reg
0268 FB                 MOV     A,PhzR30        ;get Phz data
0269 A0                 Mov     @TmpR00,A       ;store CR Next phase index addr
026A B478       DMExit: Call    DlyLng
026C B490               Call    DeSlSM
026E 83                 RET

                ; PG

                ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                ;       Stepper Motor Phase Shift Definitions

                ;       All program procedures call this data.
                ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

0300                    ORG     300H

                ;       DEFINE PHASE ADDRESSES:
                ;       THE PHASE DATA IS ENCODED TO THE ADDRESS CALLED DURING THE
                ;       STPR MTR ENERGIZE SEQUENCE CORRESPONDING TO THE NEXT PHASE
                ;       OF THE SEQUENCE REQUIRED.

                ;       CARRAGE MOTOR ENCODING: FORWARD  -  LEFT-to-RIGHT
                ;                               REVERSE  -  RIGHT-to-LEFT
```

```
                ;       Reverse direction ENCODING is the same bytes accessed in
                ;         reverse direction

0300 01                 DB      CRMFP1
0301 03                 DB      CRMFP2
0302 02                 DB      CRMFP3
0303 00                 DB      CRMFP4

                ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

                ;       LF MOTOR PHASE ENCODE & DECODE: FORWARD (CLOCKWISE)
                ;       Forward direction ENCODING:
                ;       ------------------------------------------------

0308                    ORG     308H

0308 04                 DB      LFMFP1
0309 0C                 DB      LFMFP2
030A 08                 DB      LFMFP3
030B 00                 DB      LFMFP4


                ; PG

                ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                ;       Accel/Decel / Character Handling Test

                ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                ;       TEST  >  Is CR Stpr Mtr At Speed ??
                ;         Yes - SetUp do Character Processing
                ;         No  - Calculate / Store the Acceleration Phase Shift Time (11)
                ; ---------------------------------------------------------------------------

030C FA         ADPTst: Mov     A,GStR20        ;get the status byte
030D B211               JB5     PHFSet          ;test if Stpr Mtr At Speed
                                                ;  jmp to Prnt Head Fire Setup
030F 4400               Jmp     ADMmTS          ;else Call Accel/Decel Memory Time Store

                ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                ;        Process Characters for Printing

                ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

                ;       Character dot matrix - normal char
                ;       d = Dot Column
                ;       b = Blank Column

                ;       b d d d d d
                ;       (Char Matrix)
                ;       0 0 0 0  b
                ;       0 0 0 1  d
                ;       0 0 1 0  d
                ;       0 0 1 1  d
                ;       0 1 0 0  d
                ;       0 1 0 1  d
                ; ---------------------------------------------------------------------------

0311 2668       PHFSet: JNTO    Retrn           ;if R=0 not ready to print-exit
0313 326A               JB1     NPRet           ;if Do Not Print stat bit set - EXIT
0315 D21B               JB6     SinkSt          ;if bit previously set-skip setting it
0317 FA                 Mov     A,GStR20        ;get the status byte
0318 4340               ORL     A,#SnkSet       ;set Prnt Ready Sink bit
031A AA                 Mov     GStR20,A        ;save the status byte
031B D5         SinkSt: SEL     RB1
031C FA                 Mov     A,ChStR1        ;get char status register addr
031D D23A               JB6     PageCk          ;test Char Init Done, 1 = Print Dot
                                                ;  0 = Get Char

                ; PG

                ; ---------------------------------------------------------------------------
                ;       Call for Individual character processing:  mid line test if CR/(LF)
                ; ---------------------------------------------------------------------------
                GetChr:
                ;       test for CR/(LF) if it is the test position in the line
031F F1         CRChCk: Mov     A,@CAdrR1       ;get character
0320 03F3               ADD     A,#CRCpl        ;test for Carriage Return
0322 C626               JZ      CrLnCk          ;  if CR go service it
0324 6437               Jmp     AsciC1          ;if not CR Insert Space Char
0326 FA         CRLnCk: Mov     A,ChStR1        ;get char status register addr
0327 F22B               JB7     HlfLn           ;test Chr Stat Byte Returned
                                                ;  if bit 7 = 1 then Print L-to-R
0329 6432               Jmp     SpFill          ;if R-to-L print skip exit upon CR detect
                ; ---------------------------------------------------------------------
```

```
                 ;       if L-to-R printing exit the line if less than 1/2 line printed
032B FD          HlfLn:  Mov     A,CCntR1        ;load char cnt reg w/char bufr size
032C 03D9                ADD     A,#HlfCpl       ;add the 2's cpl of 1/2 chr buf size
032E F632                JC      LnPad           ;if CB>1/2 full set CR/LF stat bit for pad
                                                 ;If CB<1/2 set buffer full stat bit
0330 648A                Jmp     MdLnEx          ;mid-line exit
                 SpFill:
0332 97          LnPad:  Clr     C               ;clear carry flag
0333 2320                Mov     A,#Space        ;insert a space char
0335 6438                Jmp     ChIsrt          ;char inserted jmp over get char
                 ; ---------------------------------------------------------------------
0337 F1          AsciC1: Mov     A,@CAdrR1       ;get character
0338 7498        ChIsrt: Call    GChar1          ;call the char lookup/trns table
                 ; ---------------------------------------------------------------
                 ;       fetch the char dot column data
                 PageCk:                         ;page test for balance of char
033A FA                  Mov     A,ChStR1        ;get the status byte
033B B241                JB5     FxJmp1          ;fix jmp over page boundries
033D F4EB                Call    ChrPg2          ;Ascii char 50 - 7F Hex
033F 6443                Jmp     MtxTst          ;jump to Matrix Test
0341 D4F0        FxJmp1: Call    ChrPg1          ;Ascii char 20 - 4F Hex
                                                 ;  fall thru to print matrix
                                                 ;  and CB count tests

                 ; PG
                 ; ---------------------------------------------------------------------
                 ;       test the Char dot column print matrix count and Char buffer count
                 ; ---------------------------------------------------------------------
0343 EB61        MtxTst: DJNZ    CDtCR1,PrntDt   ;test for dot col or blank
                                                 ;status byte in A upon entry here
0345 FA                  Mov     A,ChStR1        ;get the status byte
0346 53BF                ANL     A,#CIntND       ;set Char Init NotDone stat Flag
0348 AA                  Mov     ChStR1,A        ;store the status byte
0349 ED58                DJNZ    CCntR1,NotLCh   ;dec char cnt-jmp if Not Last Char
034B 53FD                ANL     A,#NCBFln       ;if 0 reset stat bit Not CB Full Line
034D 53FE                ANL     A,#ClICBR       ;reset CB Reg Init Flag - do Init
034F AA                  Mov     ChStR1,A        ;save the status byte

0350 C5                  SEL     RB0
0351 FA                  Mov     A,GStR20        ;get Gen Status register addr
0352 53FE                ANL     A,#NotRdy       ;clear the ready bit
0354 AA                  Mov     GStR20,A        ;store the General Status Byte
0355 D5                  SEL     RB1
0356 6468                Jmp     Retrn           ;EXit

                 ;       Test for L-to-R (forward) or R-to-L (reverse) printing
                 ;         (see GChar1 ASCII char code translation procedure)
                 ; ---------------------------------------------------------------------
                 NotLCh:                         ;A contains LR/RL bit properly set
0358 FA                  Mov     A,ChStR1        ;get char status register addr
0359 F25E                JB7     StpCh2          ;test Chr Stat Byte Returned
                                                 ;  if bit 7 = 1 then Print L-to-R
035B 19          StpCh1: Inc     CAdrR1          ;Increment char data memory addr.
035C 6468                Jmp     Retrn
035E C9          StpCh2: Dec     CAdrR1          ;Decrement char data memory addr.
035F 6468                Jmp     Retrn           ;  fall thru to Get Char

                 ; ---------------------------------------------------------------------
                 ;       Re-Entry Exit point for same char:
                 ;         (before returning step the matrix)
                 ; ---------------------------------------------------------------------
                 ;       Test for L-to-R (forward) or R-to-L (reverse) printing
                 ;         (see GChar1 ASCII char code translation procedure)
                 ; ---------------------------------------------------------------------

                 PrntDt:
0361 FA          PrnDir: Mov     A,ChStR1        ;get char status byte
0362 F267                JB7     StpCD2          ;test Chr Stat Byte Returned
                                                 ;  if bit 7 = 1 then Print L-to-R
0364 CC          StpCD1: Dec     CDotR1          ;reverse step char dot col index
                                                 ;  addr if R-to-L print
0365 6468                Jmp     Retrn           ;skip over L-to-R print addr inc
0367 1C          StpCD2: INC     CDotR1          ;forward step char dot col index
                                                 ;  addr if L-to-R print
                                                 ;EXIT

                 ; PG
```

```
                ; ------------------------------------------------------------------
                ;       Character Print SetUp Exit Procedures
                ; ------------------------------------------------------------------
                ;       Clean Standard Exit
                ; ------------------------------------------------------------------
0368 C5         Retrn:  SEL     RB0
0369 83                 Ret                     ;EXIT - return w/ Reg Bank 0 Reset

                ;       Do Not Print exit:  set Stpr Mtr drive routine count loop
036A D5         NPRet:  SEL     RB1
036B FA                 Mov     A,ChStR1        ;get the status byte
036C F27C               JB7     SkpNPI          ;test print direction
                ; Reverse
036E C5                 SEL     RB0
036F FA                 Mov     A,GStR20        ;get the status byte
0370 53BF               ANL     A,#ClrSnk       ;reset the print ready bit- skips PHFire call
0372 83                 Ret
                ; Forward
0373 D27C               JB6     SkpNPI          ;test for first PHFSet entry reg init
                ;       Initialize register variables upon first entry
                ;         end of count clears char to print bit in status byte
0375 4340               ORL     A,#ChIntD       ;set Char Reg Init Done stat bit
0377 AA                 Mov     ChStR1,A        ;save the status byte
0378 B807               Mov     TmpR10,#07H     ;load CR stpr mtr count during NoPrnt
037A 6488               Jmp     NPExit
037C E888       SkpNPI: DJNZ    TmpR10,NPExit
037E FA                 Mov     A,ChStR1        ;get the status byte
037F 53BF               ANL     A,#CIntND       ;reset - char init not done
0381 AA                 Mov     ChStR1,A        ;save the status byte
0382 C5                 SEL     RB0
0383 FA                 Mov     A,GStR20        ;get Gen Status register addr
0384 53FE               ANL     A,#NotRdy       ;clear the ready bit
0386 AA                 Mov     GStR20,A        ;store the General Status Byte
0387 83         NSetEx: Ret
0388 C5         NPExit: SEL     RB0
0389 83                 Ret

                ;       Mid-Line Exit
                ; ------------------------------------------------------------------
                ;       EXIT - if CR and not > 1/2 line done during L-to-R print
038A FA         MdLnEx: Mov     A,ChStR1        ;get the status byte
038B 53FD               ANL     A,#NCBFln       ;if 0 reset stat bit Not CB Full Line
038D 53FE               ANL     A,#ClICBR       ;reset CB Reg Init Flag - do Init
038F AA                 Mov     ChStR1,A        ;save the status byte
0390 C5                 SEL     RB0
0391 FA                 Mov     A,GStR20        ;get the RB0 status byte
0392 4302               ORL     A,#DoNotP       ;set the Do Not Print Flag(for RAccel)
0394 53BF               ANL     A,#ClrSnk       ;reset the print ready bit-exit FAccel
0396 AA                 Mov     GStR20,A        ;save the status byte
0397 83                 Ret

                ; PG
                ; ------------------------------------------------------------
                ;       Character Dot Generator Math
                ;         Look-up Table Page Vectoring
                ;         Print Head Firing
                ; ------------------------------------------------------------

0398 AE         GCHAR1: MOV     StrCR1,A                ;STORE THE CHAR

                ;  screen for printable char [char +(cpl 20 Hex + 1 = E0 Hex)]
0399 03E0               ADD     A,#0E0H
039B F69F               JC      PrntCh
039D 64C9               jmp     CntlCh          ;jmp to control char lookup table
039F 97         PrntCh: Clr     C               ;clear carry flag
03A0 FE                 Mov     A,StrCR1        ;get the char again

                ;  screen for char page [char +(cpl 50 Hex + 1 = B0 Hex)]
                ;    if carry char on page 2 else page 1
03A1 03B0               ADD     A,#0B0H
03A3 F6AE               JC      Page2

                ;       Page 1 Character -- ASCII 20 Hex thru 4F Hex
                ;         Correct offset for lookup table page
                ;         {(char + E0 Hex)*5 = Page 1 index addr}
                ; ------------------------------------------------------------
03A5 FA         Page1:  Mov     A,ChStR1        ;get the status byte
03A6 4320               OrL     A,#ChOnP1       ;set the page rentry flag bit
03A8 AA                 Mov     ChStR1,A        ;store the status byte
03A9 FE                 Mov     A,StrCR1        ;get the char agian
03AA 03E0               ADD     A,#0E0H         ;set page 1 relative 00 offset
03AC 64B8               Jmp     Multi5          ;jump to address math function
```

```
                 ;       Page 2 Character -- ASCII 20 Hex thru 4F Hex
                 ;         Correct offset for lookup table page two's complement
                 ;         of ASCII chr code LookUp Table page base char of 50H plus
                 ;         char * 5 {(char + B0 Hex)*5 = Page 2 index addr}
                 ; ---------------------------------------------------------------
03AE 97          Page2:  Clr     C                   ;clear carry flag
03AF FA                  Mov     A,ChStR1            ;get the status byte
03B0 53DF                AnL     A,#ChOnP2           ;set the page rentry flag bit
03B2 AA                  Mov     ChStR1,A            ;store the status byte
03B3 FE                  Mov     A,StrCR1            ;get the char agian
03B4 03B0                ADD     A,#0B0H             ;set page 2 relative 00 offset
03B6 64B8                Jmp     Multi5              ;fall thru to address math function

                 ;       Compute character page offset dot pattern index address
03B8 AE          MULTI5: Mov     StrCR1,A            ;store the zero offset char
03B9 E7                  RL      A                   ;MULTIPLY CHR BY 5 TO
03BA E7                  RL      A                   ;  FIND THE ADDRESS
03BB 6E                  ADD     A,StrCR1            ;ADD 1 TO COMPLETE 5X
03BC AC                  MOV     CDotR1,A            ;SAVE THE ADDRESS

                 ;       Test for L-to-R (forward) or R-to-L (reverse) printing
                 ;         (see GChar1 ASCII char code translation procedure)
                 ; ---------------------------------------------------------------
03BD FA                  Mov     A,ChStR1            ;get char status byte
03BE F2C4                JB7     LRPrn               ;test Chr Stat Byte Returned
                                                     ;  if bit 7 = 1 then Print L-toR
03C0 FC                  MOV     A,CDotR1            ;get the char index addr
03C1 0304                ADD     A,#RLPShf           ;add char offset - start at end
                                                     ;of char, print it R-to-L
03C3 AC                  MOV     CDotR1,A            ;SAVE THE ADDRESS

                 ;       Set the status byte for Character SetUp done
                 ; ---------------------------------------------------------------
03C4 FA          LRPrn:  Mov     A,ChStR1            ;get the status byte
03C5 4340                ORL     A,#ChIntD           ;set 1st char col test bit = 0
03C7 AA                  Mov     ChStR1,A            ;store the status byte
03C8 83                  Ret                         ;return w/status byte in A
                 ;       test for non printable characters goes here
03C9 83          CntlCh: Ret

                 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                 ;       Print Head Fire
                 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

                 ;       Entry point for print head solenoid firing
                 ;         - test for status byte for dot/blank column position
03CA D5          PHFire: SEL     RB1
03CB FB                  Mov     A,CDtCR1            ;set the chr dot column cnt
03CC 96D2                JNZ     Fire                ;if char cnt not 0 - Fire Head Sol.
                                                     ;if Chr Dot Cnt 0, reset the
03CE BB06        SetCnt: Mov     CDtCR1,#NDtCCt      ;  char dot column count
03D0 64D8                Jmp     Retrn1              ;skip PH Fire
03D2 2340        Fire:   MOV     A,#PTrgLo           ;get the Prnt Head Trigger byte
03D4 3A                  OUTL    P2,A                ;FIRE PRINT HEAD
03D5 23C0                MOV     A,#PTrgHi           ;get the Prnt Head Trigger byte
03D7 3A                  OUTL    P2,A                ;FIRE PRINT HEAD
03D8 C5          Retrn1: SEL     RB0
03D9 83                  Ret                         ;EXIT - return w/ Reg Bank 0 Reset

                 ; PG

                 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                 ;       PaperFeed Stpr Mtr Drive
                 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

0400                     ORG     400H

                 ;       Init psuedo register with LF inderect addr start - subsequent
                 ;         exchanges of the psuedo register will yield correct value
0400 BC04        InitLF: MOV     CntR40,#ILFCNT      ;INIT PHASE COUNT REG
0402 B822                Mov     TmpR00,#LPSAdr      ;get Phz Inderect Addr psuedo reg
0404 2308                MOV     A,#StLFF            ;get LF starting addr
0406 A0                  Mov     @TmpR00,A           ;store LF phase index addr start
                                                     ;  in psuedo register
0407 BE01                Mov     LnCtR0,#LineCt      ;set line count reg for 1 ln
                                                     ; enables exit following LF SM init
0409 841B                Jmp     LfDrv1              ;jump over line/form feed amd variable
                                                     ;  line spacing tests & setups

                 ;       LineFeed / FormFeed Drive
```

```
                ; ---------------------------------------------------------------------
                ;       load step count constant for standard line spacing
                ; ---------------------------------------------------------------------
                ;       test for various line/inch spacing would go here
                ;         (and removal of constant setup below)
040B BC1B               MOV     CntR40,#LPI8p8  ;init cnt reg for standard line feed

                ;       LineFeed/FormFeed Test
040D FA         LfDriv: Mov     A,GStR20        ;get the status byte
040E 5214               JB2     FmFd            ;if linefeed jmp to cnt load
0410 BE01       LnCtLd: Mov     LnCtR0,#LineCt  ;set line count reg for 1 line
0412 841B               Jmp     LfDrv1          ;jmp to Start of Drive
0414 FE         FmFd:   Mov     A,LnCtR0        ;get the line count
0415 37                 Cpl     A               ;2's cpl Line Count
0416 0301               Add     A,#01
0418 0342               Add     A,#PgLnCt       ;Add 2's cpl for Paging
                                                ;  PgLnCt - LnCt = n Lines to move
                                                ;  PgLnCt+(cpl(LnCt) = n lines to move
041A AE                 Mov     LnCtR0,A        ;set the line count for FF

                ;       for stablization of unused stpr mtr during CR stpr mtr drive,
                ;         store the unused stpr mtr current phase bits
041B B821       LFDrv1: Mov     TmpR00,#CPSAdr   ;get the CR phz storeage addr
041D F0                 Mov     A,@TmpR00       ;get the byute stored there
041E E3                 MovP3   A,@A            ;get the phz data byte
041F B820               Mov     TmpR00,#LastPh  ;load Last Phz psuedo reg to Temp Reg
0421 A0                 Mov     @TmpR00,A       ;store Last Phase bits - inderect
                ;       exchange/store the phase register index addresses
0422 B822               Mov     TmpR00,#LPSAdr  ;get Phz Inderect Addr psuedo reg
0424 F0                 Mov     A,@TmpR00       ;get LF last phase index addr
0425 AB                 Mov     PhzR30,A        ;place last LF phase index addr in Phz Reg
0426 BD98               MOV     TConR0,#LFTMR1  ;Load time constant Reg

                ;       Select the Stpr Mtr
0428 2306               MOV     A,#SLF          ;GET CR SM SELECT BITS
042A 3D                 MOVD    P5,A            ;SELECT SM [SCR80]

                ; ---------------------------------------------------------------------
                ;       LineFeed / FormFeed Drive Loop
                ; ---------------------------------------------------------------------
042B FB                 MOV     A,PhzR30        ;get the phz reg indirect addr index
042C E3                 MovP3   A,@A            ;do indirect get of phz bits
                ;       patch together the CR last and LF next phase bits
042D B820               Mov     TmpR00,#LastPh  ;load Last Phz psuedo reg to Temp Reg
042F 40                 ORL     A,@TmpR00        ;patch together CR existing & new LF
                ;       start timer and step motor
0430 3C                 MOVD    P4,A            ;OUTPUT BITS

                StrtLF:
0431 FD         STRLFT: MOV     A,TConR0        ;get time constant from reg
0432 62                 MOV     T,A             ;load the timer
0433 55                 STRT    T               ;START TIMER
                ;       setup the next phase to output
0434 1B                 INC     PhzR30          ;STEP PHASE DB ADDRESS
0435 FB                 MOV     A,PhzR30        ;get the phase index address
0436 523A               JB2     ZROPHL          ;test phase
0438 843C               JMP     NXTPHL
043A BB08       ZROPHL: MOV     PhzR30,#STLFF   ;re-init phase register

043C FB         NXTPHL: MOV     A,PhzR30        ;get the phz reg indirect addr index
043D E3                 MovP3   A,@A            ;do indirect get of phz bits
                ;       patch together the CR last and LF next phase bits
043E B820               Mov     TmpR00,#LastPh  ;load Last Phz psuedo reg to Temp Reg
0440 40                 ORL     A,@TmpR00        ;patch together CR existing & new LF

0441 1645       TLoopL: JTF     NXPHLF          ;jmp on time out to output nxt phz
0443 8441               JMP     TLOOPL          ;loop until timer times out

0445 3C         NXPHLF: MOVD    P4,A            ;step motor - OUTPUT BITS
0446 EC31               DJNZ    CntR40,StrLFT   ;test for end of phase count for line
                                                ;prep for next line

                ;       test for various line/inch spacing would go here
0448 BC1B               MOV     CntR40,#LPI8p8  ;init cnt reg for standard line feed
044A EE31               DJNZ    LnCtR0,StrtLF   ;test for end of line count

044C FA                 Mov     A,GStR20        ;Get the status byte
044D 53FB               ANL     A,#LineFd       ;reset for line feed
044F AA                 Mov     GStR20,A        ;save the status byte

                ;       store the phase register index addresses
                ;       Set LineFeed Stpr Mtr Next Phase index address
0450 B822       SetLRN: Mov     TmpR00,#LPSAdr  ;get Phz Storage Addr psuedo reg
```

```
0452 FB          1416          Mov     A,PhzR30          ;get the phase index address
0453 A0          1417          Mov     @TmpR00,A         ;store LF Next phase index addr
0454 B478        1418          Call    DlyLng
0456 B490        1419          Call    DeS1SM
                 1420
                 1421 ;        Check if Char Buffer contains full line (80 char or nChar & CR)
                 1422 ;          exit otherwise continue to read in characters
                 1423 ;         Mov     A,GStR20          ;get the stat byte
                 1424 ;         JB1     ByPas1            ;if Do Not Print Bit Set - EXIT
                 1425 ;         Call    CBFck
0458 83          1426 ByPas1:  Ret
                 1427
                 1428 ; PG
                 1429 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                 1430 ;        Minor Software Subroutines
                 1431 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                 1432
0500             1433          ORG     500H
                 1434
                 1435 ; ---------------------------------------------------------------
                 1436 ;        System initialization subroutines
                 1437 ; ---------------------------------------------------------------
                 1438 Defalt:
                 1439 ; ---------------------------------------------------------------
                 1440 ;        reset/set EOF status flag bit = 0
0500 D5          1441          SEL     RB1
0501 FA          1442          Mov     A,ChStR1          ;get the char status byte
0502 53F7        1443          ANL     A,#ClrEOF         ;clear the EOF flag bit
0504 AA          1444          Mov     ChStR1,A          ;store the char status byte
0505 B823        1445          Mov     TmpR10,#PTAscS    ;get the Ascii code tmp store addr
0507 B020        1446          Mov     @TmpR10,#Ascii    ;load the tmp stor reg w/ascii start
0509 C5          1447          SEL     RB0
                 1448 ; ---------------------------------------------------------------
                 1449 ;        reset/set Ok-to-Print status flag bit = 0
050A FA          1450          Mov     A,GStR20          ;get the status byte
050B 53FD        1451          ANL     A,#OkPrnt         ;reset print flag - Ok Print
050D AA          1452          Mov     GStR20,A          ;save the status byte
050E 83          1453          RET
                 1454 InitAl:
                 1455 AllOff:
                 1456 ; ---------------------------------------------------------------
                 1457 ;        CLEAR all outputs
050F C5          1458          SEL     RB0
0510 230F        1459          MOV     A,#0FH            ;FORCE PORT HI - R/ OF 555
0512 3E          1460          MOVD    P6,A
0513 23FF        1461          MOV     A,#0FFH           ;TURN ALL PRNT SOL's OFF
0515 39          1462          OUTL    P1,A
0516 23C0        1463          MOV     A,#PTRGHI         ;print head fire tirgger inactive
0518 3A          1464          OUTL    P2,A
0519 BA03        1465          ORL     P2,#03            ;set comm hdsk to ACK hi/Busy hi
051B BA00        1466          Mov     GStR20,#00H       ;clear the status registers
051D D5          1467          SEL     RB1
051E BA00        1468          Mov     ChStR1,#00H
0520 C5          1469          SEL     RB0
0521 83          1470          RET                       ;RETURN TO INIT ROUTINE
                 1471
                 1472 ; PG
                 1473 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                 1474 ;        Home Carriage / Print Head Assembly
                 1475 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                 1476
0522 FA          1477 CRHome:  Mov     A,GStR20          ;get the status byte
0523 4302        1478          ORL     A,#DoNotP         ;set the do not print flag
0525 AA          1479          Mov     GStR20,A          ;save the status byte
0526 362A        1480          JTO     RtoL              ;test for position of PH assembly
                 1481                                    ;  drive accordingly
0528 3402        1482          Call    FAccel            ;drive CR Stpr Mtr
052A 3422        1483 RtoL:    Call    RAccel            ;find the logical left home CR position
                 1484                                    ;delay a long time before continuing
052C B474        1485          Call    DlyVLg
052E 83          1486          Ret
                 1487
                 1488 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                 1489 ;        Clear Data Memory
                 1490 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                 1491
                 1492 ;        At PowerUp or Reset, following CR & LF Stpr Mtr Init, this
                 1493 ;        procedure clears data memory above RB0, Stack and RB1.
052F B87F        1494 ClrDM:   MOV     R0,#DMTop         ;GET BUFFER START LOCATION [HEX]
0531 B95D        1495          MOV     R1,#DMSIZE
0533 B000        1496 ClrDM1:  MOV     @R0,#00H          ;ZERO MEMORY LOCATION
```

```
0535 C8          1497          DEC     R0
0536 E933        1498          DJNZ    R1,ClrDM1        ;dec buffer, loop if not zero[end]
0538 83          1499          RET                      ;RETURN TO INIT ROUTINE
                 1500
                 1501 ; PG
                 1502 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                 1503 ;         Character Print TEST
                 1504 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                 1505
                 1506 PrnTst:
                 1507 ;   TEST ---  load the char buffer with successive increments of
                 1508 ;             the ascii code start.  test for end of ascii
                 1509 ;             printable chars and reinit the char stream loaded.
                 1510
0539 B97F        1511 CTInt:   Mov     CAdrR1,#FCBfSt   ;load char reg w/char bufr strt
053B BD50        1512          Mov     CCntR1,#ChBfSz   ;load char cnt reg w/char bufr size
                 1513 ChTst:                            ;Test char buffer fill with ASCII Char Code
053D FF          1514          Mov     A,opnr71         ;get the ascii char
053E A1          1515          Mov     @CAdrR1,A        ;load data memory w/Char
053F C9          1516          DEC     CAdrR1           ;Decrement dat memory location
0540 1F          1517          INC     opnr71           ;Increment Ascii char number
0541 0382        1518          ADD     A,#PAsEnd        ;test for ascii code end
0543 9647        1519          JNZ     ChrTQo           ;if not end jmp over code restart
0545 BF20        1520          Mov     OpnR71,#Ascii
0547 ED3D        1521 ChrTQo:  DJNZ    CCntR1,ChTst     ;dec buffer, loop if not zero[end]
0549 C5          1522          SEL     RB0
054A 83          1523          RET                      ;ELSE RETURN TO INIT ROUTINE
                 1524
                 1525 ; PG
                 1526 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                 1527 ;         CR Stpr Mtr Power On Initialization and
                 1528 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                 1529 ;         This routine drives the CR or LF stpr mtr for four phase
                 1530 ;           shifts for initialization.
                 1531 INITCR:                           ;POWER ON INIT STPR MTR
054B BC04        1532          MOV     CntR40,#PhCnt1   ;load phase cnt reg for INIT
054D 2308        1533          MOV     A,#SCR80         ;GET CR SM SELECT BITS
054F 3D          1534          MOVD    P5,A             ;SELECT SM [SCR80]
0550 BDC0        1535          MOV     TConR0,#IntTm2   ;Load time constant Reg
0552 BB00        1536          MOV     PhzR30,#FStCRP   ;zero SM phase reg - forward
0554 FB          1537          MOV     A,PhzR30         ;get phase index register byte
0555 E3          1538          MovP3   A,@A             ;load indexed phase shift byte
0556 3C          1539          MOVD    P4,A             ;OUTPUT BITS
0557 FD          1540 STRTTR:  MOV     A,TConR0         ;GET TIMER CONSTANT
0558 62          1541          MOV     T,A
0559 55          1542          STRT    T                ;START TIMER
055A 1B          1543          INC     PhzR30           ;step phase index register
055B FB          1544          MOV     A,PhzR30         ;CHECK THE PHASE COUNT REG
055C 5260        1545          JB2     ZroRg2
055E A462        1546          JMP     NxtPhR
0560 BB00        1547 ZroRg2:  MOV     PhzR30,#FStCRP   ;zero SM phase reg - forward
                 1548 NxtPhR:
0562 FB          1549          MOV     A,PhzR30         ;get phase index register byte
0563 E3          1550          MovP3   A,@A             ;load indexed phase shift byte
0564 1669        1551 TLoopR:  JTF     NXPHR1           ;JMP ON TIME OUT TO NEXT PH
0566 A464        1552          JMP     TLoopR           ;LOOP UNTIL TIME OUT
0568 3C          1553          MOVD    P4,A             ;OUTPUT BITS
0569 EC57        1554 NXPHR1:  DJNZ    CntR40,STRTTR
                 1555 ; ---------------------------------------------------------------
                 1556 ;         store the last phase register index addresses
056B B821        1557          Mov     TmpR00,#CPSAdr   ;get Phz Storage Addr psuedo reg
056D FB          1558          Mov     A,PhzR30         ;place last CR phase index addr in Phz Reg
056E A0          1559          Mov     @TmpR00,A        ;  store CR last phase index addr
056F B478        1560          Call    DlyLng
0571 B490        1561          Call    DeS1SM
0573 83          1562          RET
                 1563
                 1564 ; PG
                 1565 ; ---------------------------------------------------------------
                 1566 ;         Time Delay Subroutines
                 1567 ; ---------------------------------------------------------------
                 1568
                 1569 ;         Very Long
0574 B87F        1570 DlyVLg:  MOV     TmpR00,#7FH      ;LOAD DELAY COUNT IN REG.
0576 A47E        1571          Jmp     DlyST
                 1572
                 1573 ;         Long
0578 B880        1574 DlyLng:  MOV     TmpR00,#DlyCL    ;LOAD DELAY COUNT IN REG.
057A A47E        1575          Jmp     DlyST
                 1576
```

```
                 1577 ;          Not So Long - Short
057C B830        1578 DlySht:    MOV     TmpR00,#DlyCS   ;LOAD DELAY COUNT IN REG.
                 1579
                 1580 ;          Start Delay
057E 23CC        1581 DlyST:     MOV     A,#DlyTim       ;GET MAX TIMER DELAY
0580 62          1582 NxtTLd:    MOV     T,A             ;LOAD TIMER
0581 55          1583            STRT    T               ;START TIMER
                 1584
0582 168D        1585 DlyLop:    JTF     DlyTO           ;LOOP
                 1586
                 1587 ;          Char buffer fill during time loop:
0584 D5          1588            SEL     RB1
0585 FA          1589            Mov     A,ChStR1        ;get the character stat reg byte
0586 928A        1590            JB4     SkpCI           ;  test for normal char input
                 1591                                    ;   or skip if char prnt test
0588 1469        1592            Call    IBFSrv          ;service the char buffer fill
058A C5          1593 SkpCI:     SEL     RB0
058B A482        1594            JMP     DlyLOP
058D E880        1595 DlyTO:     DJNZ    TmpR00,NxtTLd   ;dec delay count & test for exit
058F B3          1596            RET
                 1597 ; ---------------------------------------------------------------
                 1598 ;          Stpr Mtr Deselect
                 1599 ; ---------------------------------------------------------------
                 1600 ;          Stepper Motor DeSelect Routine
                 1601 DESLSM:                            ;DESELECT LF/CR SM
0590 230E        1602 SMEROR:    MOV     A,#SMOFF        ;GET LF/CR SM DE-SELECT BITS
0592 3D          1603            MOVD    P5,A            ;DE-SELECT CR SM
0593 B3          1604            RET
                 1605
                 1606 $INCLUDE(:F1:CHRTBL.OV1)
                =1607
                =1608 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                =1609 ;          Character Dot Generator Look-up Table Page 1
                =1610 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                =1611
                =1612
                =1613 ;                  Character Table Page 1, contains
                =1614
                =1615 ;       20H ----------------------------------------> 4FH
                =1616
                =1617 ;  " (sp)!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLM "
                =1618
                =1619 ; ---------------------------------------------------------------
                =1620
0600            =1621           ORG     600H
                =1622
                =1623 ; ---------------------------------------------------------------
                =1624 ;          Page 1  --  Character Dot Pattern Fetch
                =1625 ;          <<<  actual assembled character table code not listed  >>>
                =1626 ; ---------------------------------------------------------------
                =1627 $NoList
                =1676 $List
                =1677 ;          Listing below is for reference only, actual code is not listed
                =1678 ;            at assembly time.
                =1679
                =1680 ; ---------------------------------------------------------------
                =1681 ; asc20:          DB      7FH, 7FH, 7FH, 7FH, 7FH         ;SPACE
                =1682 ; asc21:          DB      7FH, 7FH, 20H, 7FH, 7FH         ;!
                =1683 ; asc22:          DB      7FH, 7FH, 78H, 7FH, 78H         ;"
                =1684 ; asc23:          DB      6BH, 00H, 6BH, 00H, 6BH         ;#
                =1685 ; asc24:          DB      5BH, 55H, 00H, 55H, 6DH         ;$
                =1686 ; asc25:          DB      5CH, 6CH, 77H, 1BH, 1DH         ;%
                =1687 ; asc26:          DB      19H, 26H, 26H, 59H, 2FH         ;&
                =1688 ; asc27:          DB      7FH, 7FH, 7CH, 7FH, 7FH         ;'
                =1689 ; asc28:          DB      63H, 5DH, 3EH, 7FH, 7FH         ;(
                =1690 ; asc29:          DB      7FH, 7FH, 3EH, 5DH, 63H         ;)
                =1691 ; asc2A:          DB      5DH, 6BH, 00H, 6BH, 5DH         ;*
                =1692 ; asc2B:          DB      77H, 77H, 41H, 77H, 77H         ;+
                =1693 ; asc2C:          DB      7FH, 3FH, 4FH, 7FH, 7FH         ;,
                =1694 ; asc2D:          DB      77H, 77H, 77H, 77H, 77H         ;-
                =1695 ; asc2E:          DB      7FH, 1FH, 1FH, 7FH, 7FH         ;.
                =1696 ; asc2F:          DB      5FH, 6FH, 77H, 7BH, 7DH         ;/
                =1697 ; asc30:          DB      41H, 2EH, 36H, 3AH, 41H         ;0
                =1698 ; asc31:          DB      7FH, 3DH, 00H, 3FH, 7FH         ;1
                =1699 ; asc32:          DB      3DH, 1EH, 2EH, 36H, 39H         ;2
                =1700 ; asc33:          DB      5DH, 3EH, 36H, 36H, 49H         ;3
                =1701 ; asc34:          DB      67H, 6BH, 6DH, 00H, 6FH         ;4
                =1702 ; asc35:          DB      58H, 3AH, 3AH, 3AH, 46H         ;5
                =1703 ; asc36:          DB      43H, 35H, 36H, 36H, 4EH         ;6
                =1704 ; asc37:          DB      7EH, 0EH, 76H, 7AH, 7CH         ;7
                =1705 ; asc38:          DB      49H, 36H, 36H, 36H, 49H         ;8
```

```
=1706 ; asc39:        DB      39H, 36H, 36H, 56H, 61H          ;9
=1707 ; asc3A:        DB      7FH, 7FH, 6BH, 7FH, 7FH          ;:
=1708 ; asc3B:        DB      7FH, 3FH, 4BH, 7FH, 7FH          ;;
=1709 ; asc3C:        DB      77H, 6BH, 5DH, 3EH, 7FH          ;<
=1710 ; asc3D:        DB      6BH, 6BH, 6BH, 6BH, 6BH          ;=
=1711 ; asc3E:        DB      7FH, 3EH, 5DH, 6BH, 77H          ;>
=1712 ; asc3F:        DB      79H, 7EH, 26H, 7AH, 7DH          ;?
=1713 ; asc40:        DB      41H, 3EH, 22H, 36H, 71H          ;@
=1714 ; asc41:        DB      03H, 6DH, 6EH, 6DH, 03H          ;A
=1715 ; asc42:        DB      00H, 36H, 36H, 36H, 49H          ;B
=1716 ; asc43:        DB      41H, 3EH, 3EH, 3EH, 5DH          ;C
=1717 ; asc44:        DB      00H, 3EH, 3EH, 5DH, 63H          ;D
=1718 ; asc45:        DB      00H, 36H, 36H, 36H, 36H          ;E
=1719 ; asc46:        DB      00H, 76H, 76H, 76H, 76H          ;F
=1720 ; asc47:        DB      41H, 3EH, 3EH, 2EH, 0DH          ;G
=1721 ; asc48:        DB      00H, 77H, 77H, 77H, 00H          ;H
=1722 ; asc49:        DB      7FH, 3EH, 00H, 3EH, 7FH          ;I
=1723 ; asc4A:        DB      5FH, 3FH, 3FH, 3FH, 40H          ;J
=1724 ; asc4B:        DB      00H, 77H, 6BH, 5DH, 3EH          ;K
=1725 ; asc4C:        DB      00H, 3FH, 3FH, 3FH, 3FH          ;L
=1726 ; asc4D:        DB      00H, 7DH, 73H, 7DH, 00H          ;M
=1727 ; asc4E:        DB      0aaH, 0dfH, 0efH, 0f7H, 0aaH     ;test
=1728 ; asc4F:        DB      55H, 0dfH, 0efH, 0f7H, 55H       ;test
=1729 ; asc4E:        DB      00H, 7BH, 77H, 6FH, 00H          ;N
=1730 ; asc4F:        DB      41H, 3EH, 3EH, 3EH, 41H          ;O
=1731 ; ------------------------------------------------------------------------
=1732 ;       End Page 1  --  Character Dot Pattern Fetch
=1733
=1734 ; ------------------------------------------------------------------------
=1735 ;       Character Dot Pattern Fetch
=1736 ; ------------------------------------------------------------------------
=1737
06F0 FC  =1738 ChrPg1: MOV     A,CDotR1          ;get char index address offset
06F1 A3  =1739         MOVP    A,@A              ;get column dot patern byte
=1740
=1741 ;       this bit fix necessary to not underline each character
=1742 ;         this saves fixing each bit in the look up table
=1743
06F2 4380 =1744        ORL     A,#80H            ;char bit fix
06F4 39  =1745         OutL    P1,A              ;output the dot pattern
06F5 83  =1746         RET                       ;exit with byte in acc
=1747
=1748 ; ------------------------------------------------------------------------
=1749 ;       END Page 1  --  Character Dot Pattern Fetch
=1750 ; ------------------------------------------------------------------------
=1751
=1752
=1753 ;       PAGE 2  -- Character Dot Generator Look-Up Table
=1754 ; ------------------------------------------------------------------------
=1755
=1756 ;               Character Table Page 2, contains
=1757
=1758 ;     50H ----------------------------------------> 7EH
=1759
=1760 ;   " NOPQRSTUVWXYZ[\]^_(?)abcdefghijklmnopqrstuvwxyz{|}~ "
=1761
=1762 ; ------------------------------------------------------------------------
=1763
0700     =1764         ORG     700H
=1765
=1766 ; ------------------------------------------------------------------------
=1767 ;       Page 2  --  Character Dot Pattern Fetch
=1768 ;       <<<  Actual assembled character table code not listed  >>>
=1769 ; ------------------------------------------------------------------------
=1770 $NoLIST
=1818 $List
=1819 ;       Listing below is for reference only, actual code is not listed
=1820 ;         at assembly time.
=1821
=1822 ; ------------------------------------------------------------------------
=1823 ; asc50:        DB      00H, 76H, 76H, 76H, 79H          ;P
=1824 ; asc51:        DB      41H, 3EH, 2EH, 5EH, 21H          ;Q
=1825 ; asc52:        DB      00H, 76H, 66H, 56H, 39H          ;R
=1826 ; asc53:        DB      59H, 36H, 36H, 36H, 4DH          ;S
=1827 ; asc54:        DB      7EH, 7EH, 00H, 7EH, 7EH          ;T
=1828 ; asc55:        DB      40H, 3FH, 3FH, 3FH, 40H          ;U
=1829 ; asc56:        DB      60H, 5FH, 3FH, 5FH, 60H          ;V
=1830 ; asc57:        DB      00H, 5FH, 67H, 5FH, 00H          ;W
=1831 ; asc58:        DB      1CH, 6BH, 77H, 6BH, 1CH          ;X
=1832 ; asc59:        DB      7CH, 7BH, 07H, 7BH, 7CH          ;Y
```

```
=1833 ; asc5A:        DB    1EH, 2EH, 36H, 3AH, 3CH          ;Z
=1834 ; asc5B:        DB    00H, 3EH, 3EH, 3EH, 7FH          ;[
=1835 ; asc5C:        DB    7DH, 7BH, 77H, 6FH, 5FH          ;\
=1836 ; asc5D:        DB    7FH, 3EH, 3EH, 3EH, 00H          ;]
=1837 ; asc5E:        DB    6FH, 77H, 7BH, 77H, 6FH          ;^
=1838 ; asc5F:        DB    3FH, 3FH, 3FH, 3FH, 3FH          ;_
=1839 ; asc60:        DB    7DH, 7BH, 77H, 0FFH, 0FFH        ;\
=1840 ; asc61:        DB    0DFH, 0ABH, 0ABH, 0ABH, 087H     ;a
=1841 ; asc62:        DB    080H, 0B7H, 0B7H, 0B7H, 0CFH     ;b
=1842 ; asc63:        DB    0C7H, 0BBH, 0BBH, 0BBH, 0BBH     ;c
=1843 ; asc64:        DB    0CFH, 0B7H, 0B7H, 0B7H, 080H     ;d
=1844 ; asc65:        DB    0C7H, 0ABH, 0ABH, 0ABH, 0B7H     ;e
=1845 ; asc66:        DB    0F7H, 081H, 0F6H, 0FEH, 0FDH     ;f
=1846 ; asc67:        DB    0F7H, 0ABH, 0ABH, 0ABH, 0C3H     ;g
=1847 ; asc68:        DB    080H, 0F7H, 0FBH, 0FBH, 087H     ;h
=1848 ; asc69:        DB    0FFH, 0BFH, 0BBH, 0BFH, 0FFH     ;i
=1849 ; asc6A:        DB    0DFH, 0BFH, 0BBH, 0C2H, 0FFH     ;j
=1850 ; asc6B:        DB    0FFH, 080H, 0EFH, 0D7H, 0BBH     ;k
=1851 ; asc6C:        DB    0FFH, 0BEH, 080H, 0BFH, 0FFH     ;l
=1852 ; asc6D:        DB    087H, 0FBH, 0E7H, 0FBH, 087H     ;m
=1853 ; asc6E:        DB    083H, 0F7H, 0FBH, 0FBH, 087H     ;n
=1854 ; asc6F:        DB    0C7H, 0BBH, 0BBH, 0BBH, 0C7H     ;o
=1855 ; asc70:        DB    084H, 0EBH, 0EBH, 0EBH, 0F7H     ;p
=1856 ; asc71:        DB    0F7H, 0EBH, 0EBH, 0EBH, 084H     ;q
=1857 ; asc72:        DB    0FFH, 083H, 0F7H, 0FBH, 0FBH     ;r
=1858 ; asc73:        DB    0B7H, 0ABH, 0ABH, 0ABH, 0DBH     ;s
=1859 ; asc74:        DB    0FBH, 0C1H, 0BBH, 0DFH, 0FFH     ;t
=1860 ; asc75:        DB    0C3H, 0BFH, 0BFH, 0BFH, 0C3H     ;u
=1861 ; asc76:        DB    0E3H, 0DFH, 0BFH, 0DFH, 0E3H     ;v
=1862 ; asc77:        DB    0C3H, 0BFH, 0CFH, 0BFH, 0C3H     ;w
=1863 ; asc78:        DB    0BBH, 0C7H, 0EFH, 0C7H, 0BBH     ;x
=1864 ; asc79:        DB    0FFH, 0B3H, 0AFH, 0AFH, 0C3H     ;y
=1865 ; asc7A:        DB    0BBH, 09BH, 0ABH, 0B3H, 0BBH     ;z
=1866 ; ASC7B:        DB    07FH, 077H, 049H, 03EH, 03EH     ;{
=1867 ; ASC7C:        DB    0FFH, 0FFH, 088H, 0FFH, 0FFH     ;|
=1868 ; ASC7D:        DB    03EH, 03EH, 009H, 077H, 07FH     ;}
=1869 ; ASC7E:        DB    067H, 07BH, 067H, 05FH, 067H     ;~
=1870
=1871 ; ------------------------------------------------------------
=1872 ;       Character Dot Pattern Fetch
=1873 ; ------------------------------------------------------------
=1874
07EB FC =1875 ChrPg2: MOV    A,CDotR1        ;get char index address offset
07EC A3 =1876         MOVP   A,@A            ;get column dot patern byte
        =1877
        =1878 ;       this bit fix necessary to not underline each character
        =1879 ;         this saves fixing each bit in the look up table
        =1880
07ED 4380 =1881       ORL    A,#80H          ;char bit fix
07EF 39   =1882       OutL   P1,A            ;output the dot pattern
07F0 83   =1883       RET                    ;exit with byte in acc
          1884
          1885
          1886
          1887 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
          1888 ;              Program End
          1889 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
          1890
          1891        END

ASSEMBLY COMPLETE,   NO ERRORS
```

APPENDIX B. SOFTWARE PRINTER ENHANCEMENTS

This section describes several software enhancements which could be implemented as additions to the software developed for this Application Note. Space is available for most of the items described. Approximately 5 bytes of Data Memory would be required to implement most of the features. Two bytes would be used for status flags, and two bytes for temporary data or count storage. It is possible to use less than five bytes, but this would require the duplicate use of some flags, or other Data Memory storage, which will significantly complicate the software coding and debug tasks.

Special Characters or Symbols

Dot matrix printing lends itself well to the creation of custom characters and symbols. There are two aspects to implementing special characters. First, a character look-up table, and second, additional software for dec oding and processing the special characters or symbols. Special characters might be scientific notation, mathematical symbols, unique language characters, or block and line graphics characters.

The character look-up table could be an additional page of Program Memory dedicated to the special characters, or replace part, or all, of the existing look-up tables. If an additional look-up table is used, a third page test would be needed at the beginning of the Character Translation subroutine. There is fundamentally no difference between the processing of special characters and standard ASCII printable characters. If the characters require the same 5 x 7 dot matrix, the balance of the software would remain the same. If, however, the special characters require a different matrix, or the manipulation of the matrix, the software becomes more complex.

In general, the major software modification required to implement special characters is the size of the dot matrix printed or the dot matrix configuration used. In the case of scientific characters, it would often be necessary to shift the 5 x 7 matrix pattern within the available 9 x 9 matrix. Block or line graphics characters, on-the-other-hand, would require using the entire 9 x 9 print head matrix and printing during normally blank dot columns. This would require suspending the blank column blanking mechanism implemented in this Application Note. This would be the most complex aspect of implementing special characters. It would possibly change the number of required instructions, and thus the timing between PTS detection and print head solenoid trigger firing. This could cause the dot columns to be misaligned within a printed line and between lines.

In the case of a matrix change, two approaches are possible: dynamically changing the matrix, in line, as standard ASCII characters are being printed, or isolating the special characters to a separate processing flow where special characters are handled as a unique and complete line of characters only. A discussion of in line matrix changes for special characters is beyond the scope of this Appendix. It is sufficient to say that the changes would require the conditions setting the EOLN flag, character count, and dot column count software be modified during character processing and printing.

Lower Case Descenders

The general principle of implementing lower case descenders is to shift the 5 x 7 character dot matrix within the available 9 x 9 print head solenoid matrix. Implementing lower case descenders requires two software modifications and the creation of status flag for the purpose. First, the detection of characters needing descenders and setting a dedicated status flag during the character code to dot pattern translation subroutine. Second, the character dot column data output to the print head solenoids must be shifted for each dot column of the character. At the end of the character, the flag would be reset.

Inline Control Codes

Inline control codes are two to three character sequences, which indicate special hardware conditions or software flow control and branching. The first character indicates that the control code sequence is beginning and is typically an ASCII Escape character (ESC), 1BH. Termination of the inline code sequence would be indicated by a default number of code sequence characters. This would decrease the buffer size available for characters. Full 80 character line buffering would require loading the Character Buffer with a received character as a character is removed from it and processed.

The Inline Control Code test would be performed in two places: in the Character Buffer Fill subroutine and in the Character Processing (translation) subroutine. The test would be performed in the same manner that a Carriage Return (CR) character code test is implemented. Examples are horizontal tabs and expanded or condensed character fonts. In the case of horizontal tabs, 20H (Space Character) would have to be placed in the Character Buffer for inline processing during character processing and printing. Unless fixed position tabs are used, a minimum of a nibble of Data Memory would be required to maintain a "spaces-to-tab" count. Fixed tab positions could be set via another inline control code, by default of the printer software, or through the use of external hardware switch settings. The control code method of setting the tab positions is the most desireable, but the most complex to implement.

Different Character Formats

Figure B1 illustrates three different character fonts; standard, condensed, and enlarged or expanded characters. As the the figure illustrates, condensed and

enlarged characters are variations in either the number of dots and/or the space used to print them. Thus, each character is a variation of the stepper motor and/or print head solenoid trigger timings. Figure B2 illustrates the timings required to implement the additional character printing.

In addition to the three character fonts shown, it is possible to print each in bold face by printing each dot twice per dot column position. This would require little software modification, but would require a status flag. Again, care must be used to ensure that the delay in retriggering the solenoids is precisely the same for each type of event. Without this precise timing the dot column alignment will not be accurate. The software modifications needed to implement enlarged or condensed characters is essentially the same. The carriage and print head solenoid firing software flow is the same, but the timing for each changes. For condensed characters, the step Time Constant is doubled to approximately 4.08 ms, and the solenoids are fired four times within each step time. The step rate actually becomes a multiple of the solenoid firing time, and a counter incrementing once for each solenoid firing would be needed. At the count of four, the carriage stepper motor is stepped and the counter reset.

In the case of condensed characters, PTS does not play the same roll as in standard or enlarged character printing. PTS is not used to indicate the optimum print head solenoid firing time. Solenoid firing is purely a time function for condensed characters. PTS would only be used for Failsafe protection.

Enlarged characters would require the solenoids be fired twice per dot column data, in two sequential dot columns, at the same rate as standard characters. The character dot column data and dot column count would not be incremented at each output but at every other output. A flag could be used for this purpose.

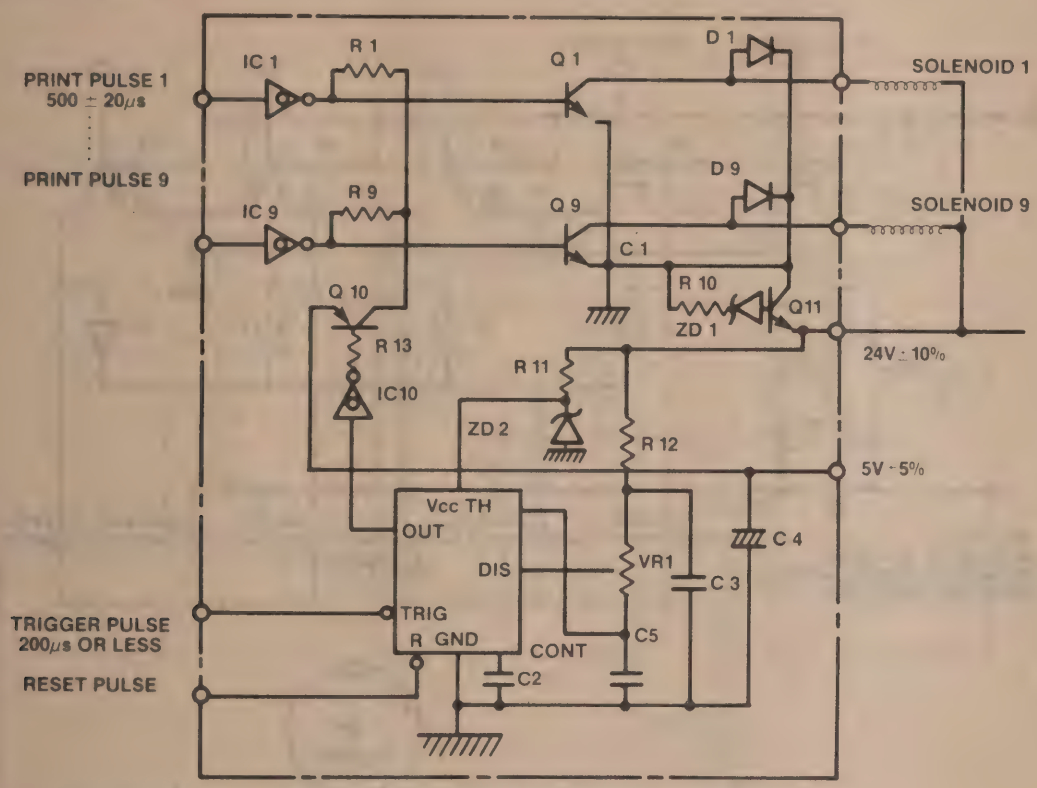
When printing either condensed or enlarged characters, the maximum character count would have to compensate for the increased or decreased characters per line count. When printing enlarged characters, the maximum characters per line would be 40. The Character Buffer could hold two complete lines of characters. But, condensed characters presents a quite different situation. The available character per line increases to 132, well beyond the 80 character Character Buffer size. The solution is to re-initialize the Character Buffer Size Count register count during condensed character processing. This will effectively inhibit the carriage stepper motor drive EOLN detection.

Two status flags would be required; one for standard or enlarged characters, and the second for condensed characters. A third status flag would be required to implement bold face printing. Activating one of the alternate character fonts could be either through the use of external status switches or through inline control code sequences, as detailed above. Note, that if the alternate character fonts are implemented in such a way that format changing is to occur dynamically during any single line being printed, the same control code problems described above also apply. In addition, the effect on the timing and dot column alignment must also be investigated.

Variable Line Spacing

Variable line spacing is another feature which could be implemented either through the use of external status switches or inline control codes. The line spacing is a function of the number of steps the stepper motor rotates for a given line. Figure 15, Paper Feed Stepper Motor Predetermined Time Constants, in the Background section above, lists the Time Constants required for three different line spacings; 6, 8, and 10 lines per inch. At the beginning of the Paper Feed Stepper Motor Drive subroutine, the default line step count is loaded. The software required is a conditional load for the line spacing, indicated by a status flag set in the External Status Switch Check subroutine or the Character Buffer Fill subroutine. Implementing the three different line spacings would require two additional status flags.

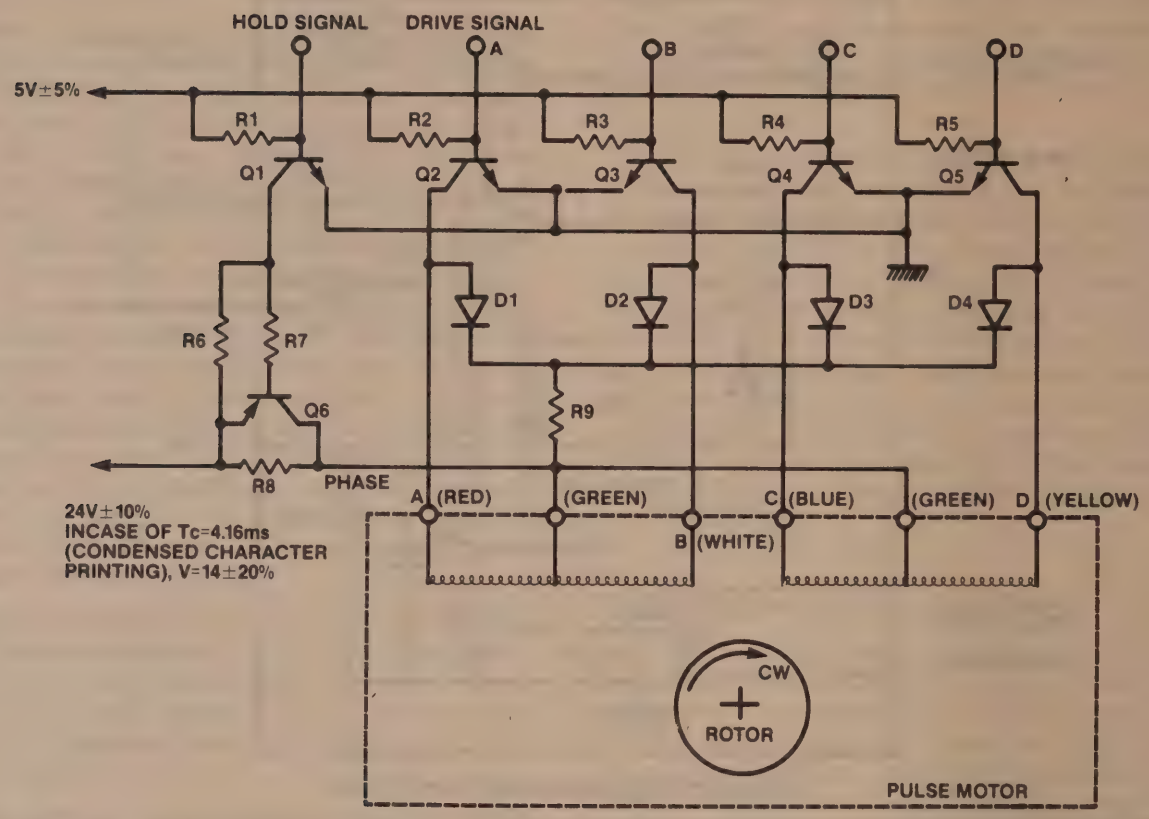
APPENDIX C. PRINTER MECHANISM DRIVE CIRCUIT



Recommended Solenoid Drive Circuit

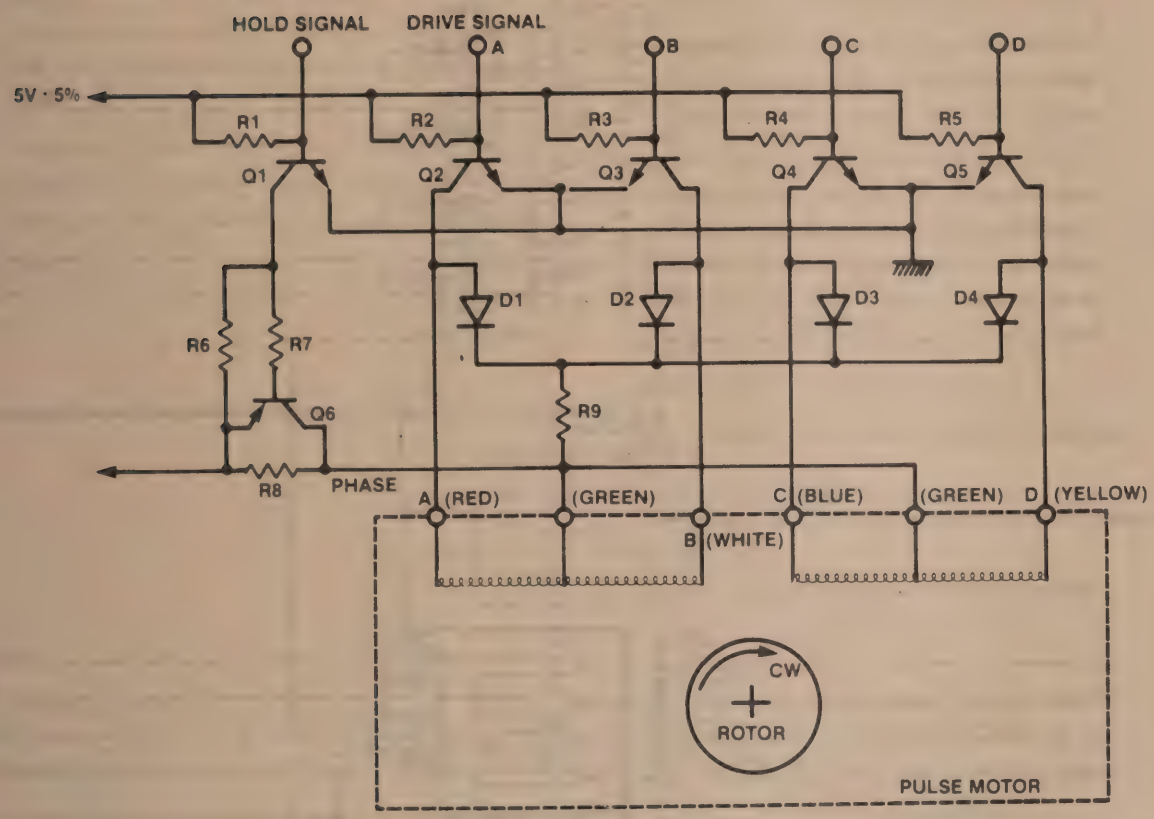
PARTS NO.		TYPE	MAKER
IC1~IC10		SN7406	TI
IC11		μA555	Fairchild
D1~D9	DIODE	S5277B	Toshiba
Q1~Q9	TRANSISTOR	2SD986	NEC
Q10	TRANSISTOR	2SA1015	Toshiba
Q11	TRANSISTOR	2SD633	Toshiba
R1~R9	RESISTOR	1.2kΩ ¼	
R10	RESISTOR	22Ω ¼	
R11	RESISTOR	580Ω 2	
R12	RESISTOR	15kΩ ¼ Carbon fil=	
R13	RESISTOR	1.2kΩ ¼	
VR1	VARIABLE RESISTOR	20kΩ ¼	
C1	CAPACITOR	1μF 100V	
C2	CAPACITOR	0.01μF	
C3	CAPACITOR	0.001μF	
C4	CAPACITOR	10μF 16V	
C5	CAPACITOR	0.1μF fil=	
ZD1	ZENOR DIODE	HZ24	Hitachi
ZD2	ZENOR DIODE	HZ5C1	Hitachi

Recommended Carriage Motor Drive Circuit

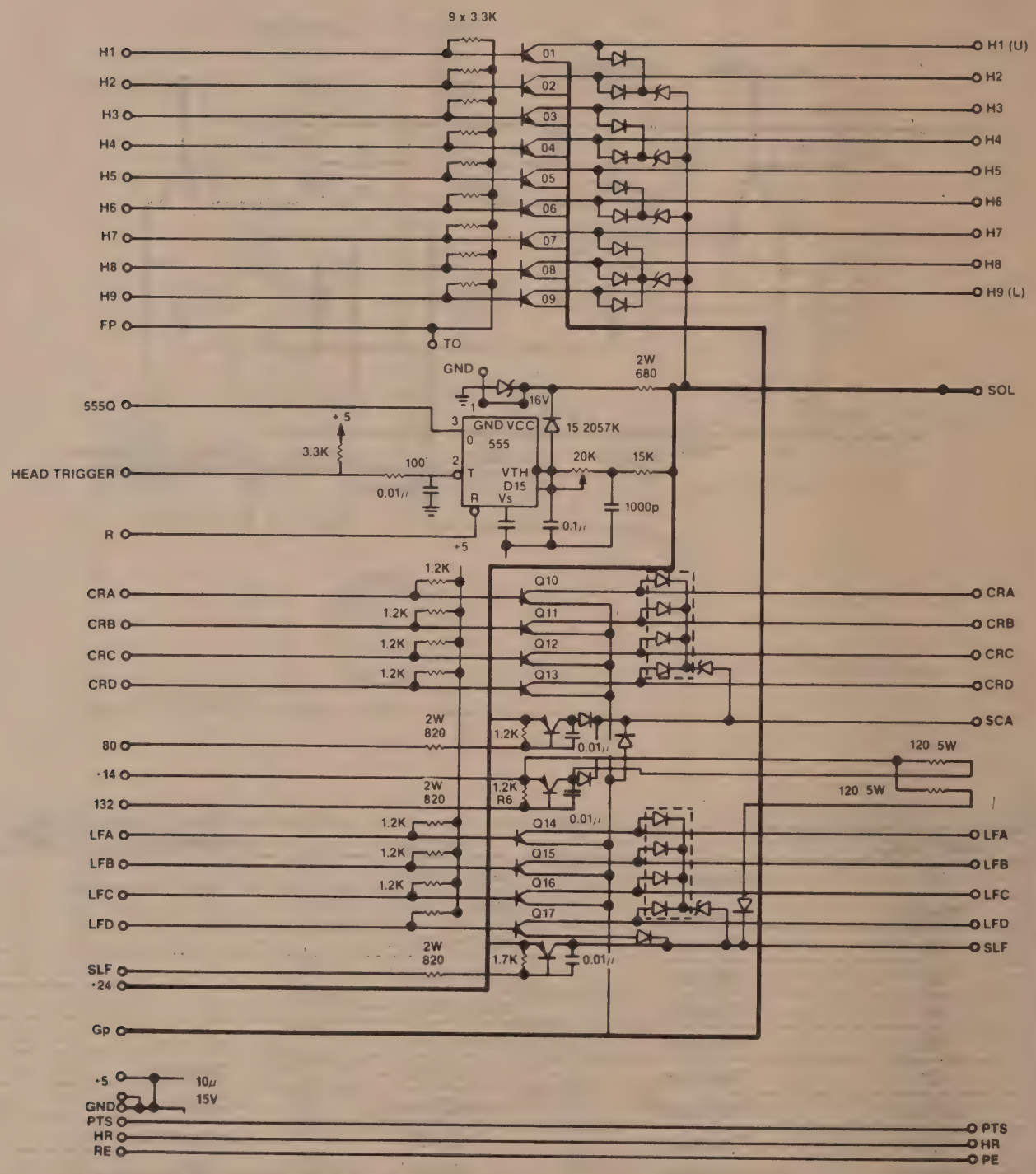


PARTS NO.		TYPE	MAKER	QTY
R1	Resistor	1kΩ±10% ¼		1
R2-R5	Resistor	220Ω±10% ¼		4
R6	Resistor	10kΩ±10% ¼		1
R7	Resistor	470Ω±10% 3		1
R8	Resistor	130Ω±10% 7		1
R9	Resistor	330Ω±10% 3		1
Q1	Transistor	2SC1815	Toshiba	1
Q2~Q5	Transistor	2SD526—Y	Toshiba	4
Q6	Transistor	2SB669	Matsushita	1
D1~D4	Diode	1S954	NEC	4

Recommended Paper Feed Motor Drive Circuit



PARTS NO.		TYPE	MAKER	QTY
R1	Resistor	1kΩ±10% ¼		1
R2-R5	Resistor	220Ω±10% ¼		4
R6	Resistor	10kΩ±10% ¼		1
R7	Resistor	470Ω±10% 3		1
R8	Resistor	130Ω±10% 7		1
R9	Resistor	330Ω±10% 3		1
Q1	Transistor	2SC1815	Toshiba	1
Q2~Q5	Transistor	2SD526—Y	Toshiba	4
Q6	Transistor	2SB669	Matsushita	1
D1~D4	Diode	1S954	NEC	4


9 x 3.3K
H1
H2
H3
H4
H5
H6
H7
H8
H9
FP
TO
01
02
03
04
05
06
07
08
09
H1 (U)
H2
H3
H4
H5
H6
H7
H8
H9 (L)
GND
2W
680
SOL
555Q
HEAD TRIGGER
R
+5
3.3K
100
0.01μ
GND VCC
555
T
R
VTH
D15
Vs
16V
15 2057K
20K
15K
1000p
0.1μ
CRA
CRB
CRC
CRD
1.2K
Q10
Q11
Q12
Q13
SCA
80
·14
132
2W
820
1.2K
R6
0.01μ
120 5W
LFA
LFB
LFC
LFD
SLF
Q14
Q15
Q16
Q17
1.7K
·24
Gp
·5
GND
PTS
HR
RE
10μ
15V
PE

intel

APPLICATION NOTES

AP-90

May 1980

An 8741A/8041A Digital Cassette Controller

John Beaston, Jim Kahn
Peripheral Applications

231314-001

INTRODUCTION

The microcomputer system designer requiring a low-cost, non-volatile storage medium has a difficult choice. His options have been either relatively expensive, as with floppy discs and bubble memories, or non-transportable, like battery backed-up RAMs. The full-sized digital cassette option was open but many times it too was too expensive for the application. Filling this void of low-cost storage is the recently developed digital mini-cassette. These mini-cassettes are similar to, but not compatible with, dictation cassettes. The mini-cassette transports are inexpensive (well under $100 in quantity), small (less than 25 cu. in.), low-power (one watt), and their storage capacity is a respectable 200K bytes of unformatted data on a 100-foot tape. These characteristics make the mini-cassette perfect for applications ranging from remote datalogging to program storage for hobbyist systems.

The only problem associated with mini-cassette drives is controlling them. While these drives are relatively easy to interface to a microcomputer system, via a parallel I/O port, they can quickly overburden a CPU if other concurrent or critical real-time I/O is required. The cleanest and probably the least expensive solution in terms of development cost is to use a dedicated single-chip controller. However, a quick search through the literature turns up no controllers compatible with these new transports. What to do? Enter the UPI-41A family of Universal Peripheral Interfaces.

The UPI-41A family is a group of two user-programmable slave microcomputers plus a companion I/O expander. The 8741A is the "flag-chip" of the line. It is a complete microcomputer with 1024 bytes of EPROM program memory, 64 bytes of RAM data memory, 16 individually programmable I/O lines, an 8-bit event counter and timer, and a complete slave peripheral interface with two interrupts and Direct Memory Access (DMA) control. The 8041A is the masked ROM, pin compatible version of the 8741A. Figure 2 shows a block diagram common to both parts. The 8243 I/O port expander completes the family. Each 8243 provides 16 programmable I/O lines.

Using the UPI concept, the designer can develop a custom peripheral control processor for his particular I/O problem. The designer simply develops his peripheral control algorithm using the UPI-41A assembly language and programs the EPROM of

Figure 1. Comparison of Mini-Cassette and Floppy Disk Transports and Media.

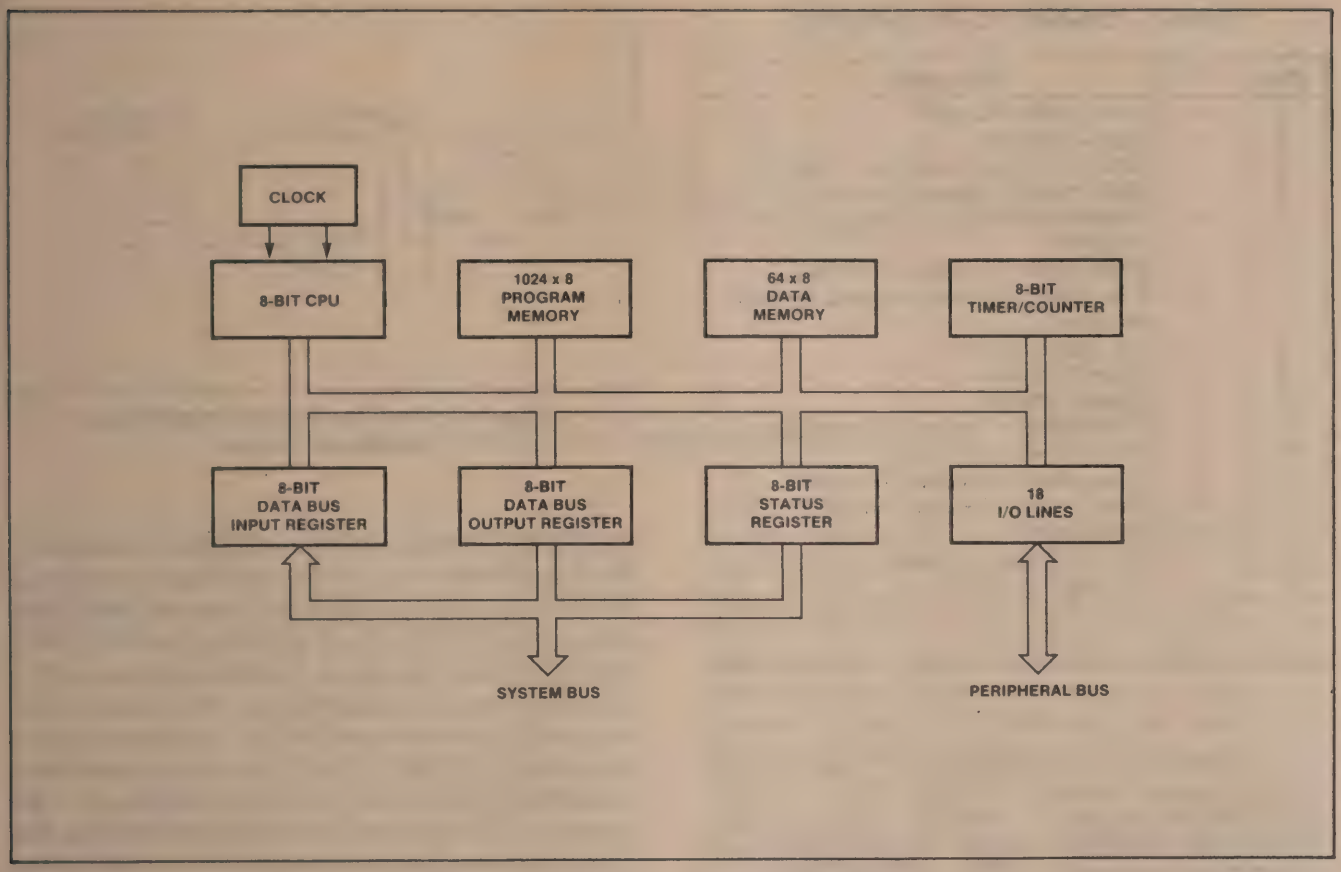


Figure 2. 8741A/8041A Block Diagram

the 8741A. Voila! He has a single-chip dedicated controller. Testing may be accomplished using either an ICE-41A or the Single-step mode of the 8741A. UPI-41A peripheral interfaces are being used to control printers, keyboards, displays, custom serial interfaces, and data encryption units. Of course, the UPI family is perfect for developing a dedicated controller for digital mini-cassette transports. To illustrate this application for the UPI family let's consider the job of controlling the Braemar CM-600 Mini-Dek®.

THE CM-600 MINI-DEK*

The Braemar CM-600 is representative of digital mini-cassette transports. It is a single-head, single-motor transport which operates entirely from a single 5-volt power supply. Its power requirements, including the motor, are 200ma for read or write and 700ma for rewind. Tapes speeds are 3 inches per second (IPS) during read or write, 5 IPS fast forward, and 15 IPS rewind. With these speeds and a maximum recording density of 800 bits per inch (BPI), the maximum data rate is 2400 bits per second (BAUD). The data capacity using both sides of a 100-foot tape is 200K bytes. On top of this, the transport occupies only 22.5 cubic inches (3″x3″x2.5″).

All I/O for the CM-600 is TTL-compatible and can be divided into three groups: motor control, data control, and cassette status. The motor group controls are GO/STOP, FAST/SLOW, and FORWARD/REVERSE. The data controls are READ/ WRITE, DATA IN, and DATA OUT. The remaining group of outputs give the transport's status: CLEAR LEADER, CASSETTE PRESENCE, FILE PROTECT, and SIDE SENSOR. These signals, shown schematically in figure 3 and table 1, give the pin definition of the CM-600 16-pin I/O connector.

RECORDING FORMAT

The CM-600 does not provide either encoding or decoding of the recorded data; that task is left for the peripheral interface. A multitude of encoding techniques from which the user may choose are available. In this single-chip dedicated controller application, a "self-clocking" phase encoding scheme similar to that used in floppy discs was chosen. This scheme specifies that a logic "0" is a bit cell with no transition; a cell with a transition is a logic "1."

Table 1. CM-600* I/O Pin Definition

Pin	I/O	Function
1	—	Index pin—not used
2	—	Signal ground
3	O	Cassette side (0—side B, 1—side A)
4	I	Data input (0—space, 1—mark)
5	O	Cassette presence (0—cassette, 1—no cassette)
6	I	Read/Write (0—read, 1—write)
7	O	File protect (0—tab present, 1—tab removed)
8	—	+5v motor power
9	—	Power ground
10	—	Chassis ground
11	I	Direction (0—forward, 1—rewind)
12	I	Speed (0—fast, 1—slow)
13	O	Data output (0—space, 1—mark)
14	O	Clear leader (0—clear leader, 1—off clear leader)
15	I	Motion (0—go, 1—stop)
16	—	+5v logic power

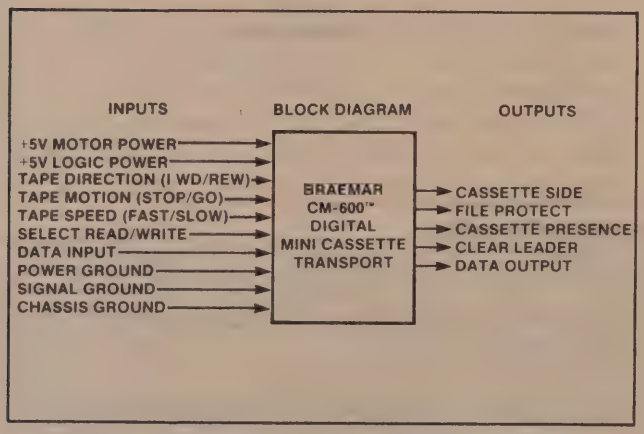


Figure 3. Braemar CM-600* Block Diagram

Figure 4 illustrates the encoding of the character 3AH assuming the previous data ended with the data line high. (The least significant bit is sent first.) Notice that there is always a "clocking" transition at the beginning of each cell. Decoding is simply a matter of triggering on this "clocking" transition, waiting 3/4 of a bit cell time, and determining whether a mid-cell transition has occurred. Cells with no mid-cell transitions are data 0's; cells with transitions are data 1's. This encoding technique has all the benefits of Manchester encoding with the added advantage that the encoded data may be "decoded by eyeball:" long cells are always 0's, short cells are always 1's.

Besides the encoding scheme, the data format is also up to the user. This controller uses a variable byte length, checksum protected block format. Every block starts and ends with a SYNC character

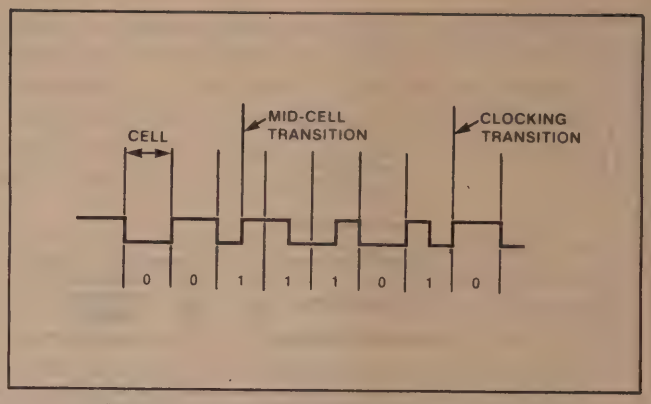


Figure 4. Modified Phase Encoding of Character 3A Hex

(AAH), and the character immediately preceeding the last SYNC is the checksum. The checksum is capable of catching 2 bit errors. The number of data characters within a block is limited to 64K bytes. Blocks are separated by an Inter-Record Gap (IRG). The IRG is of such a length that the transport can stop and start within an IRG, as illustrated in the data block timing, figure 5. Braemar specifies a maximum start or stop time of 150ms for the transport, thus the controller uses 450ms for the IRG. This gives plenty of margin for controlling the transport and also for detecting IRGs while skipping blocks.

THE UPI-41A™ CONTROLLER

The goal of the UPI software design for this application was to make the UPI-41A microcomputer into an intelligent cassette control processor. The host processor (8086, 8088, 8085A, etc.) simply issues a high-level command such as READ-a-block or WRITE-a-block. The 8741A accepts the command, performs the requested operation, and returns to the host system a result code telling the outcome of the operation, eg. Good-Completion, Sync Error, etc. Table 2 shows the command and result code repertoire. The 8741A completely manages all the data transfers for reading and writing.

As an example, consider the WRITE-a-block command. When this command is issued, the UPI-41A expects a 16-bit number from the host telling how many data bytes to write (up to 64K bytes per block). Once this number is supplied in the form of two bytes, the host is free to perform other tasks; a bit in the UPI's STATUS register or an interrupt output will notify the host when a data transfer is required. The 8741A then checks the transport's status to be sure that a cassette is present and not file protected. If either is false, a result code is

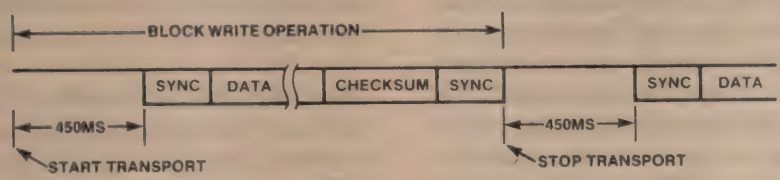


Figure 5. IRG/Block Timing Diagram (not to scale)

Table 2. Controller Command/Result Code Set

Command	Result
Read (01H)	Good-Completion (00H) Buffer Overrun Error (41H) Bad Synch1 Error (42H) Bad Synch2 Error (43H) Checksum Error (44H) Command Error (45H) End of Tape Error (46H)
Rewind (04H)	Good-Completion (00H)
Skip (03H)	Good-Completion (00H) End of Tape Error (47H) Beginning of Tape Error (48H)
Write (02H)	Good-Completion (00H) Buffer Underrun Error (81H) Command Error (82H) End of Tape Error (83H)

returned to the host; otherwise the transport is started. After the peripheral controller checks to make sure that the tape is off the clear leader and past the hole in the tape, it writes a 450ms IRG, a SYNC character, the block of data, the checksum, and the final SYNC character. (The tape has a clear leader at both ends and a small hole 6 inches from the end of each leader.) The data transfers from the host to the UPI-41A slave microcomputer are double buffered. The controller requests only the desired number of data bytes by keeping track of the count internally.

If nothing unusual happened, such as finding clear leader while writing, it returns a Good-Completion result code to the host. If clear leader was encountered, the transport is stopped immediately and an End-of-Tape result code is returned to the host. Another possible error would be if the host is late in supplying data. If this occurs, the controller writes an IRG, stops the drive, and returns the appropriate Data-Underrun result code.

The READ-a-block command also provides error checking. Once this command is issued by the host, the controller checks for cassette presence. If present, it starts the transport. The data output from the transport is then examined and decoded continuously. If the first character is not a SYNC, that's an error and the controller returns a Bad-First-SYNC result code (42H) after advancing to the next IRG. If the SYNC is good, the succeeding characters are read into an on-chip 30 character circular buffer. This continues until an IRG is encountered. When this occurs, the transport is stopped. The controller then tests that the last character. If it is a SYNC, the controller then compares the accumulated internal checksum to the block's checksum, the second to the last character of the block. If they match, a Good-Completion result code (00H) is returned to the host. If either test is bad, the appropriate error result code is returned. The READ command also checks for the End-of-Tape (EOT) clear leader and returns the appropriate error result code if it is found before the read operation is complete.

The 30 character circular buffer allows the host up to 30 character times of response time before the host must collect the data. All data transfers take place thru the UPI-41A Data Bus Buffer Output register (DBBOUT). The controller continually monitors the status of this register and moves characters from the circular buffer to the register whenever it is empty.

The SKIP-n-blocks command allows the host to skip the transport forward or backward up to 127 blocks. Once the command is issued, the controller expects one data byte specifying the number of

blocks to skip. The most significant bit of this byte selects the direction of the skip (0=forward, 1=reverse). SKIP is a dual-speed operation in the forward direction. If the number of blocks to skip is greater than 8, the controller uses fast-forward (5 IPS) until it is within 8 blocks of the desired location. Once within 8 blocks, the controller switches to the normal read speed (3 IPS) to allow accurate placement of the tape. The reverse skip uses only the rewind speed (15 IPS). Like the READ and WRITE commands, SKIP also checks for EOT and beginning-of-tape (BOT) depending upon the tape's direction. An error result code is returned if either is encountered before the number of blocks skipped is complete.

The REWIND command simply rewinds the tape to the BOT clear leader. The ABORT command allows the termination of any operation in progress, except a REWIND. All commands, including ABORT, always leave the tape positioned on an IRG.

THE HARDWARE INTERFACE

There's hardly any hardware design effort required for the controller and transport interface in figure 6. Since the CM-600 is TTL compatible, it connects directly to the I/O ports of the UPI controller. If the two are separated (i.e. on different PC cards), it is recommended that TTL buffers be provided.) The only external circuitry needed is an LED driver for the DRIVE ACTIVE status indicator.

The 8741A-to-host interface is equally straightforward. It has a standard asynchronous peripheral interface: 8 data lines (D_0-D_7), read (RD), write (WR), register select (AO), and chip select (CS). Thus it connects directly to an 8086, 8088, 8085A, 8080, or 8048 bus structure. Two interrupt outputs are provided for data transfer requests if the particular system is interrupt-driven. DMA transfer capability is also available. The clock input can be driven from a crystal directly or with the system clock (6MHz max). The UPI-41A clock may be asynchronous with respect to other clocks within the system.

This application was developed on an Intel iSBC 80/30 single board computer. The iSBC 80/30 is controlled by an 8085A microprocessor, contains 16K bytes of dual-ported dynamic RAM and up to 8K bytes of either EPROM or ROM. Its I/O complement consists of an 8255A Programmable Parallel Interface, an 8251A Programmable Communica-

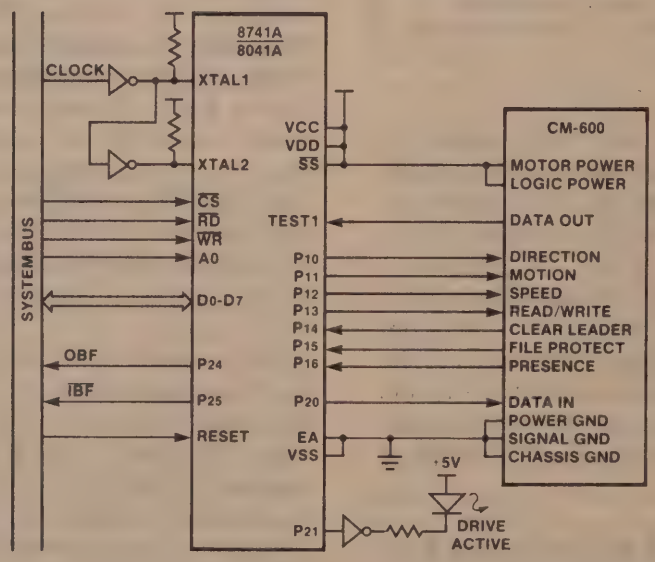


Figure 6. Controller/Transport System Schematic

tions Interface, an 8253 Programmable Interval Timer, and an 8259A Programmable Interrupt Controller. The iSBC 80/30 is especially convenient for UPI development since it contains an uncommitted socket dedicated to either an 8041A or 8741A, complete with buffering for its I/O ports. The iSBC 80/30 to 8741A interface is reflected in figure 8. (Optionally, an iSBC 569 Digital Controller board could be used. The iSBC 569 board contains three uncommitted UPI sockets with an interface similar to that in figure 8.)

Looking at the host-to-controller interface, the host sees the 8741A as three registers in the host's I/O address space: the data register, the command register, and the status register. The decoding of these registers is shown in figure 7. All data and commands for the controller are written into the Data Bus Buffer Input register (DBBIN). The state of the register select input, AO, determines whether a command or data is written. (Writes with AO set to 1 are commands by convention.) All data and results from the controller are read by the host from the Data Bus Buffer Output register (DBBOUT).

$\overline{CS}$	$\overline{RD}$	$\overline{WR}$	A0	Register
0	0	1	0	DBBOUT
0	0	1	1	STATUS
0	1	0	0	DBBIN (DATA)
0	1	0	1	DBBIN (COMMAND)
1	X	X	X	NONE

Figure 7. 8741A/8041A Interface Register Decoding

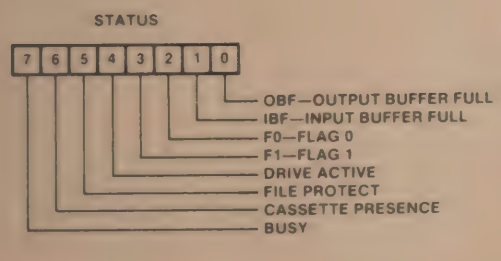


Figure 8. Status Register Bit Definition

The Status register contains flags which give the host the status of various operations within the controller. Its format is given in figure 8. The Input Buffer Full (IBF) and Output Buffer Full (OBF) flags show the Status of the DBBIN and DBBOUT registers respectively. IBF indicates when the DBBIN register contains data written by the host. The host may write to DBBIN only when IBF is 0. Likewise, the host may read DBBOUT only when OBF is set to a 1. These bits are handled automatically by the UPI-41A internal hardware. FLAG 0 (F_0) and FLAG 1 (F_1) are general purpose flags used internally by the controller which have no meaning externally.

The remaining four bits are user-definable. For this application they are DRIVE ACTIVE, FILE PROTECT, CASSETTE PRESENCE, and BUSY flags. The FILE PROTECT and CASSETTE PRESENCE flags reflect the state of the corresponding I/O lines from the transport. DRIVE ACTIVE is set whenever the transport motor is on and the controller is performing an operation. The BUSY flag indicates whether the contents of the DBBOUT register is data or a result code. The BUSY flag is set whenever a command is issued by the host and accepted by the controller. As long as BUSY is set, any character found in DBBOUT is a result code. Thus whenever the host finds OBF set, it should test the BUSY flag to determine whether the character is data or a result code.

Notice the OBF and $\overline{IBF}$ are available as interrupt outputs to the host processor, figure 6. These outputs are self-clearing, that is, OBF is set automatically upon the controller loading DBBOUT and cleared automatically by the host reading DBBOUT. Likewise $\overline{IBF}$ is cleared to a 0 by the host writing into DBBIN: set to a 1 when the controller reads DBBIN into the accumulator.

The flow charts of figure 9 show the flow of sample host software assuming a polling software interface between the host and the controller. The WRITE command requires two additional count bytes which form the 16-bit byte count. These extra bytes are "handshaked" into the controller using the IBF flag in the STATUS register. Once these bytes are written, the host writes data in response to IBF being cleared. This continues until the host finds OBF set indicating that the operation is complete and reads the result code from DBBOUT. No testing of BUSY is needed since only the result code appears in the DBBOUT register.

The READ command does require that BUSY be tested. Once the READ command is written into the

controller, the host must test BUSY whenever OBF is set to determine whether the contents of DBBOUT is data from the tape or the result code.

THE CONTROLLER SOFTWARE

The UPI-41A software to control the cassette can be divided up into various commands such as WRITE, READ and ABORT. In a previous version of this application note (May 1980), software was described that implemented these commands. This code however did not adequately compensate for speed variations of the motor during record and playback nor for data distortion caused by the magnetic media. Since then, new code has been written to include these effects. This revised software is now available through the INTEL User's Library, INSITE. For more information on this software or INSITE, contact your local INTEL Sales Office.

intel®

June 1985

Applications Using the 8042 UPI™ Microcontroller

1. The 8042 in the IBM PC/AT
2. Using the 8042 vs. using microcontrollers
3. Custom serial protocol with the 8042

Joe Froelich

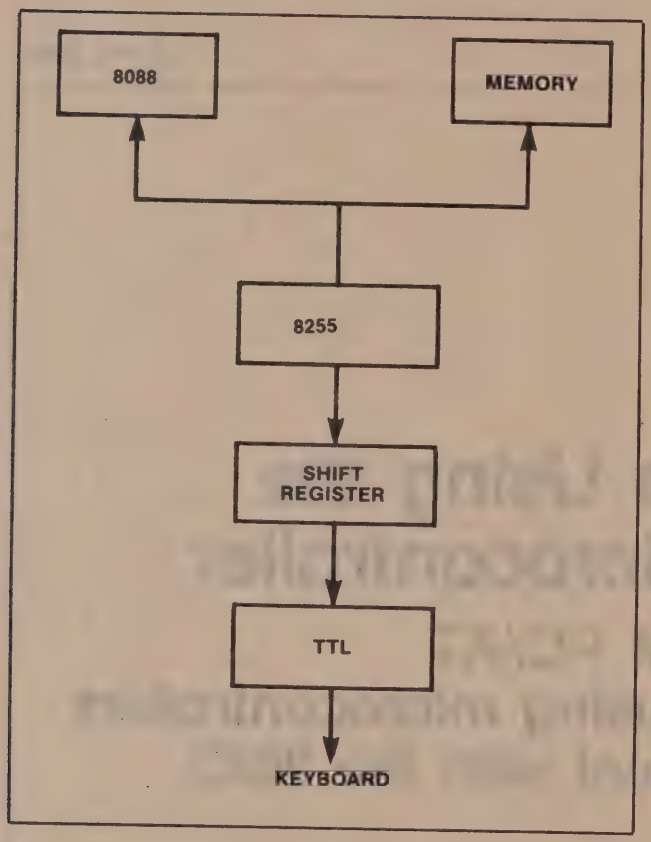
©Intel Corporation, 1985

Order Number: 231600-001

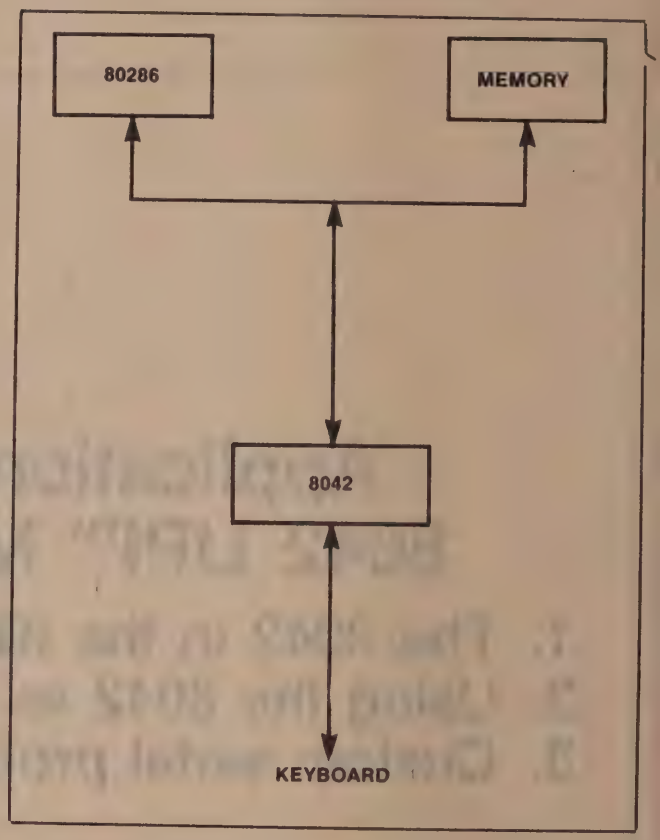
APPLICATION #1
THE 8042 UPI™ MICROCONTROLLER IN THE IBM PC/AT

The following example is an important application of UPIs but there are many more. It is truly a universal device that can be *customized* to all those "non-standard" peripheral control problems. Applications are limited only by imagination. Think UPIs for non-standard peripherals!!

IBM PC/AT (BEFORE) . . .



. . . IBM PC/AT (AFTER)



NEW FUNCTIONS

The 8042 also brings new functions to the PC/AT:

- Keyboard lockup security (front panel key)
- CRT type input to the system
- Diagnostics/self testing of keyboard interface
- Parity check and retry
- PC and PC/AT keyboard interchangeability
- Reset CPU to compatible mode
- Address wrap-around protect in compatible mode

THE FUTURE IS THE KEY

Modifications and upgrades are easy because of the 8042's programmable flexibility and power:

- Change keyboard scan codes (in 8042 ROM)
- Increase functionality of keyboard interface through software and/or unused I/O lines on 8042
- Control other PC/AT functions with these I/O lines

Summary

In short, IBM used the 8042 since it:

- Offloads housekeeping details from the CPU
- Facilitates modular system design
- Offers a customized peripheral
- Provides a clean, efficient upgrade path

These benefits can apply to many of your applications also.

APPLICATION #2
USING THE 8042 VS. USING MICROCONTROLLERS

PROBLEM

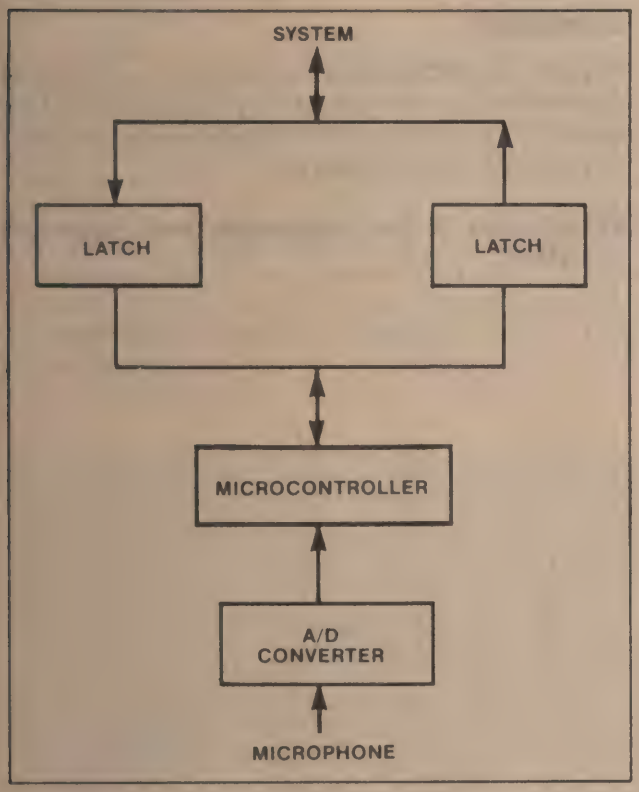
What do you do when you're making SBX and VME modules for a voice digitizing board and you need:

1) an interface to an A/D Converter with
2) 12 MHz operation,
3) an absolute minimum chip count, and
4) very low cost (for the PC market).

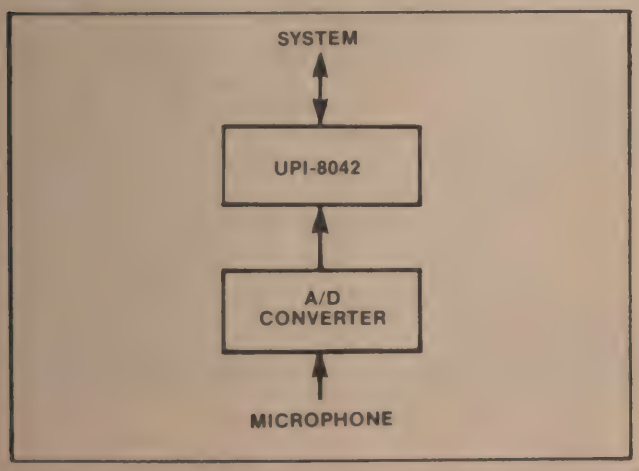
A leading vendor was faced with exactly this problem. Here is what they started with, and the bottom figure shows how they improved things with the 8042 UPI™ microcontroller.

SOLUTION

BEFORE . . .



AFTER . . .



The 8042 integrates two latches and the microcontroller into a single-chip solution.

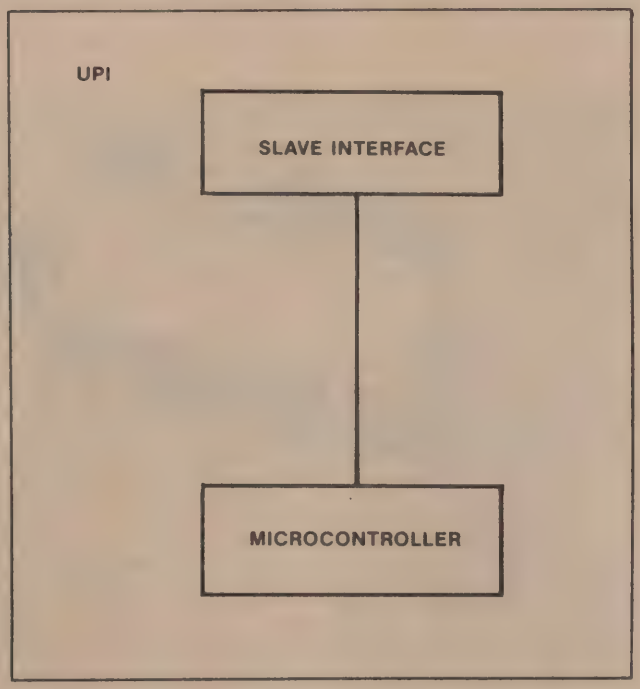
WHY THE SWITCH

After studying the four requirements for this module, it is easy to see why they switched. The first two (A/D interface and 12 MHz) were met by both solutions. However, it is clear the second alternative is much better on board space and on overall cost. There are fewer chips, so they could avoid a multi-layer board and thus save a lot in total cost. Actual chip costs are within 10% of each other (a typical microcontroller like a Z8 or 6801 plus 2 latches compared to an 8042), and they do the same thing.

WHAT'S THE DIFFERENCE

People tend to think of microcontrollers whenever there is a "non-standard" device to control. CRTs, disk drives and DRAMs all have dedicated controllers, but printers, front panels, displays and keyboards don't, because they are all "non-standard" devices. Microcontrollers can be customized to these applications.

The problem is when the device is a "slave" or a peripheral, regular microcontrollers need the extra circuitry shown previously. That's why we invented UPIs. They are simply microcontrollers with the slave interface built in. They are, therefore, more efficient to use in peripheral-type configurations.



UPIs may be misunderstood because of the funny name. They shouldn't be. It's really simple. When faced with non-standard device control, think microcontrollers.

If it's a master-only configuration, think regular microcontrollers. If it's a slave/peripheral configuration, think UPIs.

APPLICATION #3
CUSTOM SERIAL PROTOCOL WITH THE 8042 UPI™ MICROCONTROLLER

BACKGROUND

The 8042 UPI Microcontroller, because of its programmability is being used everywhere, and here is another example. A leading board vendor was designing a communications concentrator board. They wanted a way to:

1) interface 8 serial channels to a minicomputer bus
2) operate these at 4800 baud
3) use one board
4) provide a custom serial protocol that
   - communicated commands and data packets
   - assembled the data packets
   - provided handshaking signals
   - checked for framing, timing, parity, noise, modem and synchronization errors
   - provided self-test diagnostics

THE 8042 SOLUTION

There certainly wasn't an "off-the-shelf" chip that would satisfy the above requirements, and using the main CPU would have caused tremendous system performance degradation. They needed all of these features to offer a competitive product, so they looked to the 8042 UPI Microcontroller. Since the speed requirements were not too great (4800 baud), they could implement the protocol in software. The 8042's programmability gave them all the flexibility needed to incorporate the formatting, handshaking and error checking desired. Moreover, the on-chip slave interface made communication with the minicomputer's bus a snap.

SUMMARY

In short, the 8042 allowed this company to implement a custom serial communication protocol that in turn allowed them to offer a customized, competitive interface board to their customers.

Don't some of your applications need customized interfaces?

ICE™-42
8042 IN-CIRCUIT EMULATOR

- **Precise, full-speed, real-time emulation**
  - **Load, drive, timing characteristics**
  - **Full-speed program RAM**
  - **Parallel ports**
  - **Data Bus**
- **User-specified breakpoints**
- **Execution trace**
  - **User-specified qualifier registers**
  - **Conditional trigger**
  - **Symbolic groupings and display**
  - **Instruction and frame modes**
- **Emulation timer**
- **Full symbolic debugging**
- **Single-line assembly and disassembly for program instruction changes**
- **Macro commands and conditional block constructs for automated debugging sessions**
- **HELP facility: ICE™-42 command syntax reference at the console**
- **User confidence test of ICE™-42 hardware**

The ICE™-42 module resides in the Intellec Microcomputer Development System and interfaces to any user-designed 8042 or 8041A system through a cable terminating in an 8042 emulator microprocessor and a pin-compatible plug. The emulator processor, together with 2K bytes of user program RAM located in the ICE-42 buffer box, replaces the 8042 device in the user system while maintaining the 8042 electrical and timing characteristics. Powerful Intellec debugging functions are thus extended into the user system. Using the ICE-42 module, the designer can emulate the system's 8042 chip in real-time or single-step mode. Breakpoints allow the user to stop emulation on user-specified conditions, and a trace qualifier feature allows the conditional collection of 1000 frames of trace data. Using the single-line 8042 assembler the user may alter program memory using the 8042 assembler mnemonics and symbolic references, without leaving the emulator environment. Frequently used command sequences can be combined into compound commands and identified as macros with user-defined names.

©INTEL CORPORATION, 1983

MAY 1983
ORDER NUMBER: 210818-002

FUNCTIONAL DESCRIPTION

Integrated Hardware and Software Development

The ICE-42 emulator allows hardware and software development to proceed interactively. This approach is more effective than the traditional method of independent hardware and software development followed by system integration. With the ICE-42 module, prototype hardware can be added to the system as it is designed. Software and hardware integration occurs while the product is being developed. Figure 1 shows the ICE-42 emulator connected to a user prototype.

The ICE-42 emulator assists four stages of development:

SOFTWARE DEBUGGING

This emulator operates without being connected to the user's system before any of the user's hardware is available. In this stage ICE-42 debugging capabilities can be used in conjunction with the Intellec text editor and 8042 macroassembler to facilitate program development.

HARDWARE DEVELOPMENT

The ICE-42 module's precise emulation characteristics and full-speed program RAM make it a valuable tool for debugging hardware.

SYSTEM INTEGRATION

Integration of software and hardware begins when any functional element of the user system hardware is connected to the 8042 socket. As each section of the user's hardware is completed, it is added to the prototype. Thus, each section of the hardware and software is "system" tested in real-time operation as it becomes available.

SYSTEM TEST

When the user's prototype is complete, it is tested with the final version of the user system software. The ICE-42 module is then used for real-time emulation of the 8042 chip to debug the system as a completed unit.

The final product verification test may be performed using the 8742 EPROM version of the 8042 microcomputer. Thus, the ICE-42 module provides the ability to debug a prototype or production system at any stage in its development without introducing extraneous hardware or software test tools.

Symbolic Debugging

The ICE-42 emulator permits the user to define and to use symbolic, rather than absolute, references to program and data memory addresses. Thus, there is no need to recall or look up the addresses of key locations in the program, or to become involved with machine code.

When a symbol is used for memory reference in an ICE-42 emulator command, the emulator supplies the corresponding location as stored in the ICE-42 emulator symbol table. This table can be loaded with the symbol table produced by the assembler during application program assembly. The user obtains the symbol table during software preparation simply by using the "DEBUG" switch in the 8042 macroassembler. Furthermore, the user interactively modifies the emulator symbol table by adding new symbols or changing or deleting old ones. This feature provides great flexibility in debugging and minimizes the need to work with hexadecimal values.

Through symbolic references in combination with other features of the emulator, the user can easily:

- Interpret the results of emulation activity collected during trace.
- Disassemble program memory to mnemonics, or assemble mnemonic instructions to executable code.
- Reference labels or addresses defined in a user program.

Automated Debugging and Testing

MACRO COMMAND

A macro is a set of commands given a name. A group of commands executed frequently can be defined as a macro. The user executes the group of commands by typing a colon followed by the macro name. Up to ten parameters may be passed to the macro.

Macro commands can be defined at the beginning of a debug session and then used throughout the whole session. One or more macro definitions can be saved on diskette for later use. The Intellec text editor may be used to edit the macro file. The macro definitions are easy to include in any later emulation session.

The power of the development system can be applied to manufacturing testing as well as development by writing test sequences as macros. The macros are stored on diskettes for use during system test.

COMPOUND COMMAND

Compound commands provide conditional execution of commands (IF command) and execution of commands repeatedly until certain conditions are met (COUNT, REPEAT commands).

Compound commands may be nested any number of times, and may be used in macro commands.

Example:

```
*DEFINE .I=0          ; Define symbol .I to 0
*COUNT 100H           ; Repeat the following
                        commands 100H times.
.*IF .I AND 1 THEN    ; Check if .I is odd
..*CBYTE.I=.I         ; Fill the memory at
                        location .I to value .I
..*END
.*.I-.I+1             ; Increment .I by 1.
.*END                 ; Command executes
                        upon carriage-return
                        after END
```

(The asterisks are system prompts; the dots indicate the nesting level of compound commands.)

Operating Modes

The ICE-42 software is an Intellec RAM-based program that provides easy-to-use commands for initiating emulation, defining breakpoints, controlling trace data collection, and displaying and controlling system parameters. ICE-42 commands are configured with a broad range of modifiers that provide maximum flexibility in describing the operation to be performed.

EMULATION

The ICE-42 module can emulate the operation of prototype 8042 system, at real-time speed (up to 12MHz) or in single steps. Emulation commands to the ICE-42 module control the process of setting up, running, and halting an emulation of the user's 8042-based system. Breakpoints and tracepoints enable the ICE-42 emulator to halt emulation and provide a detailed trace of execution in any part of the user's program. A summary of the emulation commands is shown in Table 1.

Table 1 Major Emulation Commands

Command	Description
GO	Begins real-time emulation and optionally specifies break conditions.
BR0, BR1, BR	Sets or displays either or both Breakpoint Registers used for stopping real-time emulation.
STEP	Performs single-step emulation.
QR0, QR1	Specifies match conditions for qualified trace.
TR	Specifies or displays trace-data collection conditions and optionally sets Qualifier Register (QR0, QR1).
Synchronization Line Commands	Sets and displays status of synchronization line outputs or latched inputs. Used to allow real-time emulation or trace to start and stop synchronously with external events.

Breakpoints

The ICE-42 hardware includes two breakpoint registers that allow halting of emulation when specified conditions are met. The emulator continuously compares the values stored in the breakpoint registers with the status of specified address, opcode, operand, or port values, and halts emulation when this comparison is satisfied. When an instruction initiates a break, that instruction is executed completely before the break takes place. The ICE-42 emulator then regains control of the console and enters the interrogation mode. With the breakpoint feature, the user can request an emulation break when the program:

- Executes an instruction at a specific address or within a range of addresses.

- Executes a particular opcode.
- Receives a specific signal on a port pin.
- Fetches a particular operand from the user program memory.
- Fetches an operand from a specific address in program memory.

Trace and Tracepoints

Tracing is used with real-time and single-step emulation to record diagnostic information in the trace buffer as a program is executed. The information collected includes opcodes executed, port values, and memory addresses. The ICE-42 emulator collects 1000 frames of trace data.

If desired this information can be displayed as assembler instruction mnemonics for analysis during interrogation or single-step mode. The trace-collection facility may be set to run conditionally or unconditionally. Two unique trace qualifier registers, specified in the same way as breakpoint registers, govern conditional trace activity. The qualifiers can be used to condition trace data collection to take place as follows:

- Under all conditions (forever).
- Only while the trace qualifier is satisfied.
- For the frames or instructions preceding the time when a trace qualifier is first satisfied (pre-trigger trace).
- For the frames or instructions after a trace qualifier is first satisfied (post-triggered trace).

Table 2 shows an example of trace display.

INTERROGATION AND UTILITY

Interrogation and utility commands give convenient access to detailed information about the

Table 2 Trace Display (Instruction Mode)

FRAME	LOC	OBJ	INSTRUCTION	P1	P2	T0	T1	DBYIN	YOUT	YSTS	TOVF
0000:	100H	2355	MOV A,#55H	FFH	FFH	0	0	66H	DFH	02H	0
0004:	102H	39	OUTL P1,A	FFH	FFH	0	0	66H	DFH	02H	0
0008:	103H	3A	OUTL P2,A	55H	FFH	0	0	66H	DFH	02H	0
0012:	104H	22	IN A,DBB	55H	55H	0	0	66H		02H	0
0014:	105H	37	CPL A	55H	55H	0	0		DFH	02H	0
0016:	106H	02	OUT DBB,A	55H	55H	0	0	66H		00H	0
0018:	107H	BA03	MOV R2,#03H	55H	55H	0	0	66H	99H	00H	0
0022:	109H	B840	MOV R0,#.TABLE0	55H	55H	0	0	66H	99H	01H	0
0026:	10BH	B960	MOV R1,#.TABLE1	55H	55H	0	0	66H	99H	01H	0
.LOOP											
0030:	10DH	F0	MOV A,@R0	55H	55H	0	0		99H	01H	0
0032:	10EH	A1	MOV @R1,A	55H	55H	0	0	66H		01H	0
0034:	10FH	18	INC R0	55H	55H	0	0		99H	01H	0
0036:	110H	19	INC R1	55H	55H	0	0	66H		01H	0
0038:	111H	EA0D	DJNZ R2,.LOOP	55H	55H	0	0	66H	99H	01H	0
.LOOP											
0042:	10DH	F0	MOV A,@R0	55H	55H	0	0		99H	01H	0
0044:	10EH	A1	MOV @R1,A	55H	55H	0	0	66H		01H	0
0046:	10FH	18	INC R0	55H	55H	0	0		99H	01H	0
0048:	110H	19	INC R1	55H	55H	0	0	66H		01H	0
0050:	111H	EA0D	DJNZ R2,.LOOP	55H	55H	0	0	66H	99H	01H	0
.LOOP											
0054:	10DH	F0	MOV A,@R0	55H	55H	0	0		99H	01H	0
0056:	10EH	A1	MOV @R1,A	55H	55H	0	0	66H		01H	0
0058:	10FH	18	INC R0	55H	55H	0	0		99H	01H	0
0060:	110H	19	INC R1	55H	55H	0	0	66H		01H	0
0062:	111H	EA0D	DJNZ R2,.LOOP	55H	55H	0	0	66H	99H	01H	0
0066:	113H	00	NOP	55H	55H	0	0		99H	01H	0

user program and the state of the 8042 that is useful in debugging hardware and software. Changes can be made in memory and in the 8042 registers, flags, and port values. Commands are also provided for various utility operations such as loading and saving program files, defining symbols, displaying trace data, controlling system synchronization and returning control to ISIS-II. A summary of the basic interrogation and utility commands is shown in Table 3. Two additional time-saving emulator features are discussed below.

Single-Line Assembler/Disassembler

The single-line assembler/disassembler (ASM and DASM commands) permits the designer to examine and alter program memory using assembly language mnemonics, without leaving the emulator environment or requiring time-consuming program reassembly. When assembling new mnemonic instructions into program memory, previously defined symbolic references (from the original program assembly, or subsequently defined during the emulation session)

Table 3 Major Interrogation and Utility Commands

Command	Description
HELP	Displays help messages for ICE-42 emulator command-entry assistance.
LOAD	Loads user object program (8042 code) into user-program memory, and user symbols into ICE-42 emulator symbol table.
SAVE	Saves ICE-42 emulator symbol table and/or user object program in ISIS-II hexadecimal file.
LIST	Copies all emulator console input and output to ISIS-II file.
EXIT	Terminates ICE-42 emulator operation.
DEFINE	Defines ICE-42 emulator symbol or macro.
REMOVE	Removes ICE-42 emulator symbol or macro.
ASM	Assembles mnemonic instructions into user-program memory.
DASM	Disassembles and displays user-program memory contents.
Change/Display Commands	Change or display value of symbolic reference in ICE-42 emulator symbol table, contents of key-word references (including registers, I/O ports, and status flags), or memory references.
EVALUATE	Evaluates expression and displays resulting value.
MACRO	Displays ICE-42 macro or macros.
INTERRUPT	Displays contents for the Data Bus and timer interrupt registers.
SECONDS	Displays contents of emulation timer, in microseconds.
Trace Commands	Position trace buffer pointer and select format for trace display.
PRINT	Displays trace data pointed to by trace buffer pointer.
MODE	Sets or displays the emulation mode, 8041A or 8042.

Table 4 HELP Command

```
*HELP
Help is available for the following items. Type HELP followed by the item name.
The help items cannot be abbreviated. (For more information, type HELP HELP or
HELP INFO.)
Emulation:   Trace Collection:          Misc:        <address>
GO GR SYO    TR  QR  QRO  QR1  SY1      BASE         <CPU#keyword>
BR BROBR1                               DISABLE      <expr>
STEP         Trace Display:             ENABLE       <ICE42#keyword>
             TRACE  MOVE   PRINT        ERROR        <identifier>
             OLDEST NEWEST              EVALUATE     <instruction>
                                        HELP         <masked#constant>
Change/       Display/ Define/ Remove:  INFO         <match#cond>
<CHANGE>      REMOVE   CBYTE            <LIGHTS>     <numeric#constant>
<DISPLAY>     SYMBOL   DBYTE   DASM     LIST         <partition>
REGISTER      RESET            ASM      LOAD         <string>
                                        MODE
SECONDS       WRITE                     SAVE         <string#constant>
DEFINE        STACK            SY       SUFFIX       <symbolic#ref>
                                        SYMBOLIC     <mode>
Macro:              Compound                         <trace#reference>
DEFINE       DIR    Commands:                        <unlimited#match#cond>
DISABLE      ENABLE COUNT                            <user#symbols>
INCLUDE      PUT    IF
<MACRO#DISPLAY>      REPEAT
<MACRO#INVOCATION>
*
*
*HELP IF
IF - The conditional command allows conditional execution of one or more commands
based on the values of boolean conditions.
     IF <expr> 'THEN  <cr>              <true#list>::='<command> <cr> @
      <true#list>                       <false#list>;;='<command> <cr> @
      'ORIF <expr> <cr>                 <command>::=An ICE-42 command.
      <true#list> @
      'ELSE <cr>
      <false#list>
      END
The <expr>s are evaluated in order as 16-bit unsigned integers. If one is
reached whose value has low-order bit 1 (TRUE), all commands in the <true#list>
following that <expr> are then executed and all commands in the other <true#-
list>s and in the <false#list> are skipped. If all <expr>s have value with low-
order bit 0 (FALSE), then all commands in all <true#list>s are skipped and, if
ELSE is present, all commands in the <false#list> are executed.
     (EX:  IF .LOOP=5 THEN
             STEP
             ELSE
             GO
             END)
*
*
*
*
*EXIT
```

may be used in the instruction operand field. The emulator supplies the absolute address or data values as stored in the emulator symbol table. These features eliminate user time spent translating to and from machine code and searching for absolute addresses, with a corresponding reduction in transcription errors.

HELP

The HELP file allows display of ICE-42 command syntax information at the Intellec console. By typing "HELP", a listing of all items for which help messages are available is displayed. Typing "HELP <Item>" then displays relevant information about the item requested, including typical usage examples. Table 4 shows some sample HELP messages.

EMULATION ACCURACY

The speed and interface demands of a high-performance single-chip microcomputer require extremely accurate emulation, including full-speed, real-time operation with the full function of the microcomputer. The ICE-42 module achieves accurate emulation with an 8042 emulator chip, a special configuration of the 8042 microcomputer family, as its emulation processor.

Each of the 40 pins on the user plug is connected directly to the corresponding 8042 pin on the emulator chip. Thus the user system sees the emulator as an 8042 microcomputer at the 8042 socket. The resulting characteristics provide extremely accurate emulation of the 8042 including speed, timing characteristics, load and drive values, and crystal operation. However, the emulator may draw more power from the user system than a standard 8042 family device.

Additional emulator processor pins provide signals such as internal address, data, clock, and control lines to the emulator buffer box. These signals let static RAM in the buffer box substitute for on-chip program ROM or EPROM. The emulator chip also gives the ICE module "back-door" access to internal chip operation, allowing the emulator to break and trace execution without interfering with the values on the user-system pins.

Figure 1 A Typical 8042 Development Configuration. The host system is an Intellec Series IV. The ICE-42 module is connected to a user prototype system.

SPECIFICATIONS

ICE™-42 Operating Requirements

Intellec Model 800, Series II, Series III, or Series IV Microcomputer Development SYstem (64K RAM required)

System console (Model 800 only)

Intellec Diskette Operating System: ISIS (Version 3.4 or later).

Equipment Supplied

- Printed circuit boards (2)
- Emulation buffer box, Intellec interface cables, and user-interface cable with 8042 emulation processor
- Crystal power accessory
- Operating instructions manuals
- Diskette-based ICE-42 software (single and double density)

Emulation Clock

User's system clock (up to 12MHz) or ICE-42 crystal power accessory (12 MHz)

Environmental Characteristics

Operating Temperature — 0° to 40°C

Operating Humidity — Up to 95% relative humidity without condensation.

Physical Characteristics

Printed Circuit Boards

Width: 12.00 in. (30.48 cm)
Height: 6.75 in. (17.15 cm)
Depth: 0.50 in. (1.27 cm)

Buffer Box

Width: 8.00 in. (20.32 cm)
Length: 12.00 in. (30.48 cm)
Depth: 1.75 in. (4.44 cm)
Weight: 4.0 lb. (1.81 kg)

Electrical Characteristics

DC Power Requirements (from Intellec® system)

$V_{CC} = +5V, \pm 5\%$
$I_{CC} = 13.2A$ max; 11.0A typical
$V_{DD} = +12V, \pm 5\%$
$I_{DD} = 0.1A$ max; 0.05A typical
$V_{BB} = -10V, \pm 5\%$
$I_{BB} = 0.05A$ max; 0.01A typical

User plug characteristics at 8042 socket — Same as 8042 or 8742 except that the user system sees an added load of 25 pF capacitance and 50μA leakage from the ICE-42 emulator user plug at ports 1, 2, T0, and T1.

ORDERING INFORMATION

Part Number	Description
ICE-42	8042 Microcontroller In-Circuit Emulator, cable assembly and interactive diskette software

intel®

MCS® -48 DISKETTE-BASED SOFTWARE SUPPORT PACKAGE

- **Extends Intellec microcomputer development system to support MCS-48 development**
- **MCS-48 assembler provides conditional assembly and macro capability**
- **Takes advantage of powerful ISIS-II file handling and storage capabilities**
- **Provides assembler output in standard Intel hex format**

The MCS-48 assembler translates symbolic 8048 assembly language instructions into the appropriate machine operation codes, and provides both conditional and macroassembler programming. Output may be loaded either to an ICE-49 module for debugging or into the iUP Universal PROM Programmer for 8748 PROM programming. The MCS-48 assembler operates under the ISIS-II operating system on Intel Development systems.

© INTEL CORPORATION, 1983.

MAY 1983

Order Number: 231323-001

FUNCTIONAL DESCRIPTION

The MCS-48 assembler translates symbolic 8048 assembly language instructions into the appropriate machine operation codes. The ability to refer to program addresses with symbolic names eliminates the errors of hand translation and makes it easier to modify programs when adding or deleting instructions. Conditional assembly permits the programmer to specify which portions of the master source document should be included or deleted in variations on a basic system design, such as the code required to handle optional external devices. Macro capability allows the programmer use of a single label to define a routine. The MCS-48 assembler will assemble the code required by the reserved routine whenever the macro label is inserted in the text. Output from the assembler is in standard Intel hex format. It may be either loaded directly to an in-circuit emulator (ICE-49) module for integrated hardware/software debugging, or loaded into the iUP Universal PROM Programmer for 8748 PROM programming. A sample assembly listing is shown in Table 1.

The MCS 48 assembler supports the 8048, 8049, 8050, 8020, 8021, 8022, 8041 and 8042. The MCS 48 assembler can also support CMOS versions of the 8048 family.

Table 1. Sample MCS-48 Diskette-Based

```
ISIS-II 8048 MACROASSEMBLER, V1.0                                    PAGE 1

LOC  OBJ                SEQ          SOURCE STATEMENT

                         1   DECIMAL ADDITION ROUTINE  ADD BCD NUMBER
                         2   AT LOCATION 'BETA' TO BCD NUMBER AT 'ALPHA' WITH
                         3   RESULT IN 'ALPHA'  LENGTH OF NUMBER IS 'COUNT' DIGIT
                         4   PAIRS (ASSUME BOTH BETA AND ALPHA ARE SAME LENGTH
                         5   AND HAVE EVEN NUMBER OF DIGITS OR MSD IS 0 IF
                         6   ODD)
                         7   INIT     MACRO    AUGND,ADDND,CNT
                         8            MOV      R0, #AUGND
                         9   L1:      MOV      R1, #ADDND
                        10            MOV      R2, #CNT
                        11            ENDM
                        12
0001E                   13   ALPHA    EQU      30
0028                    14   BETA     EQU      40
0032                    15   COUNT    EQU      5
0100                    16            ORG      100H
                        17            INIT     ALPHA, BETA, COUNT
0100  B81E              18+           MOV      R0, #ALPHA
0102  B928              19+ L1:       MOV      R1, #BETA
0104  BA32              20+           MOV      R2, #COUNT
0106  97                21            CLR      C
0107  F0                22   LP:      MOV      A, @R0
0108  71                23            ADDC     A, @R1
0109  57                24            DA       A
010A  A1                25            MOV      @R0, A
010B  18                26            INC      R0
010C  19                27            INC      R1
010D  EA07              28            DJNZ     R2, LP
                                      END
USER SYMBOLS
ALPHA  0001E    BETA 0028    COUNT 0005    LP 0107
L1     0102

ASSEMBLY COMPLETE, NO ERRORS


ISIS II ASSEMBLER SYMBOL CROSS REFERENCE  V1.0                      PAGE 1

SYMBOL CROSS REFERENCE

ALPHA  13#   17
BETA   14#   17
COUNT  15#   17
INIT    7#   17
L1     19#
LP     22#   28
```

SPECIFICATIONS

Operating Environment

(All) Intel Microcomputer Development Systems
(Series II, Series III/Series IV)
Intel Personal Development System

Documentation Package

Titles of: User Guides
Operating Instructions
Reference Manuals

Ordering Information

Part Number	Description
MDS-D48*	MCS-48 Disk Based Assembler

Requires Software License

SUPPORT:

Hotline Telephone Support, Software Performance Reports (SPR), Software Updates, Technical Reports, Monthly Newsletters are available.

*MDS is an ordering code only and is not used as a product name or trademark. MDS is a registered trademark of Mohawk Data Sciences Corporation.

iUP-200A/iUP-201A UNIVERSAL PROM PROGRAMMERS

MAJOR iUP-200A/iUP-201A FEATURES:

- Support for all Intel PROM families through multiple-device personality modules, which may also be used with the Intel personal development system (iPDS™).
- Serial interface to all Intellec® development systems.
- Powerful PROM programming software (iPPS).
- iUP system self-tests plus device integrity checks.
- Support for new personality modules that provide state of the art fast programming algorithms, the inteligent Identifier™, and a security bit.

ADDITIONAL iUP-201A FEATURES:

- Off-line editing, device duplication, and PROM memory locking.
- 32K-byte iUP RAM.
- 24-character alphanumeric display.
- Full hexadecimal plus 12-function keypad.

The iUP-200A and iUP-201A universal programmers program and verify data in all the Intel programmable ROMs (PROMs). They can also program the PROM memory portions of Intel's single-chip microcomputer and peripheral devices. When used with any Intellec® development system, the iUP-200A and iUP-201A universal programmers provide on-line programming and verification using the Intel PROM programming software (iPPS). In addition, the iUP-201A universal programmer supports off-line, stand-alone program editing, PROM duplication, and PROM memory locking. The iUP-200A universal programmer is expandable to an iUP-201A model.

The following are trademarks of Intel Corporation and may be used only to describe Intel products CREDIT, Index, Intel, Insite, Intellec, Library Manager, Megachassis, Micromap, MULTIBUS, PROMPT, UPI, μScope, Promware, MCS, ICE, iRMX, iSBC, iSBX, inteligent Identifier, MULTIMODULE and ICS. Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.
© INTEL CORPORATION, 1984

March, 1984
Order Number: 210319-003

FUNCTIONAL DESCRIPTION

The iUP-200A universal programmer operates in on-line mode. The iUP-201A universal programmer operates in both on-line and off-line mode.

On-line System Hardware

The iUP-200A and iUP-201A universal programmers are free-standing units that, when connected to any Intel development system having at least 64K bytes of host memory, provide on-line PROM programming and verification of Intel programmable devices. In addition, the universal programmer can read the contents of the ROM versions of these devices.

The universal programmer communicates with the host through a standard RS-232C serial data link. A serial converter is needed when using the MDS 800 as a host system. (Serial converters are available from other manufacturers.)

Each universal programmer contains an 8085 CPU, selectable power supply, 4.3K bytes of static RAM, a programmable timer, an interface for personality modules, an interface for the host system, and 12K bytes of programmed EPROM. The iUP-201A also has a keyboard and display. The programmed EPROM contains the firmware needed for all universal programmer editing and control functions.

A personality module adapts the universal programmer to a family of PROM devices; it contains all the hardware and firmware necessary to program either a family of Intel PROMs or a single Intel device. The user inserts the personality module into the universal programmer front panel. The personality module comes ready to use; no additional sockets or adapters are required.

Figure 1 shows the iUP-200A on-line system configuration, and Figure 2 shows the on-line system data flow.

On-line System Software

The Intel PROM programming software (iPPS) is included with both the iUP-200A and iUP-201A models of the universal programmer. Created to run on any Intellec development system, the iPPS software provides user control through an easy-to-use interactive interface. The iPPS software performs the following functions to make PROM programming quick and easy:

- Reads PROMs and ROMs
- Programs PROMs directly or from a file

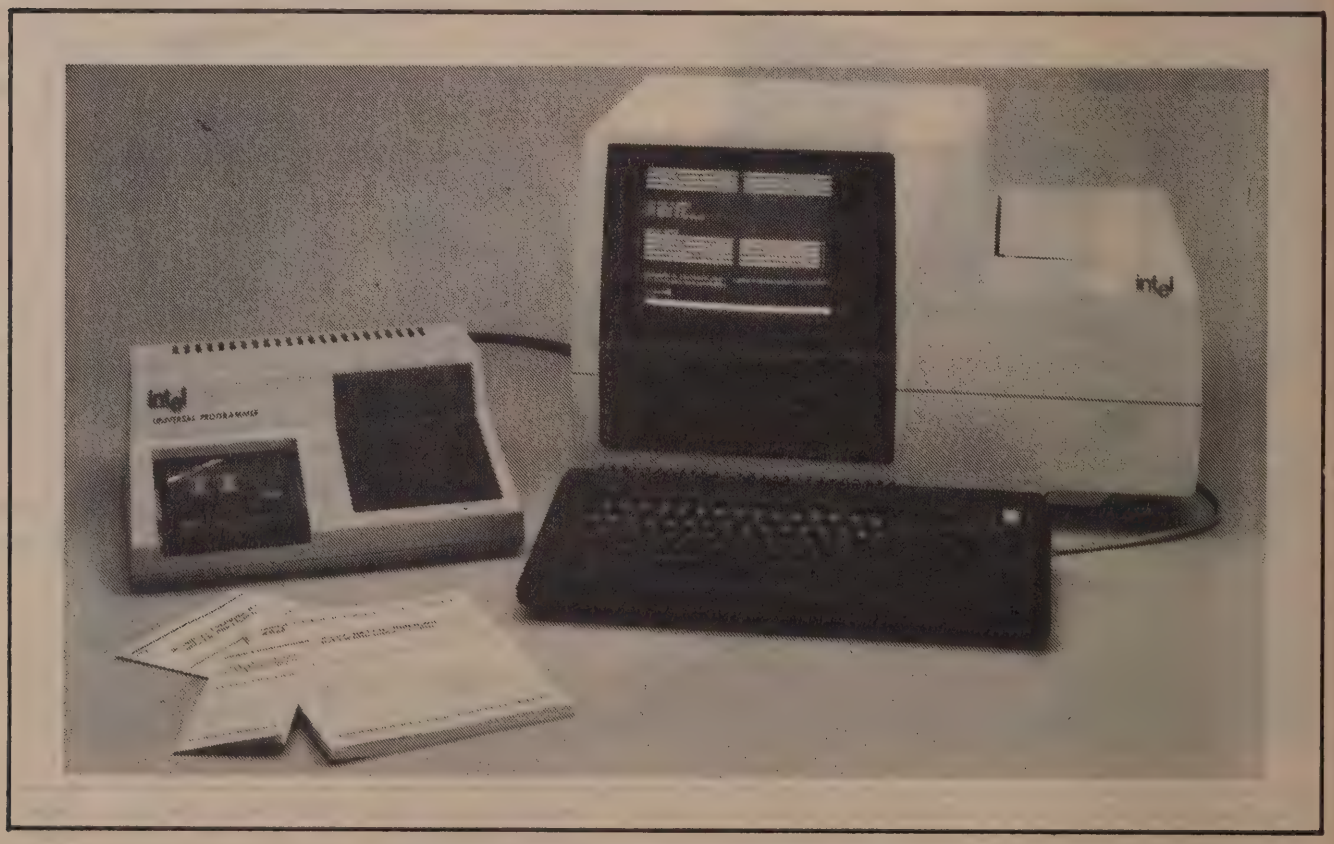


Figure 1 On-Line System Configuration

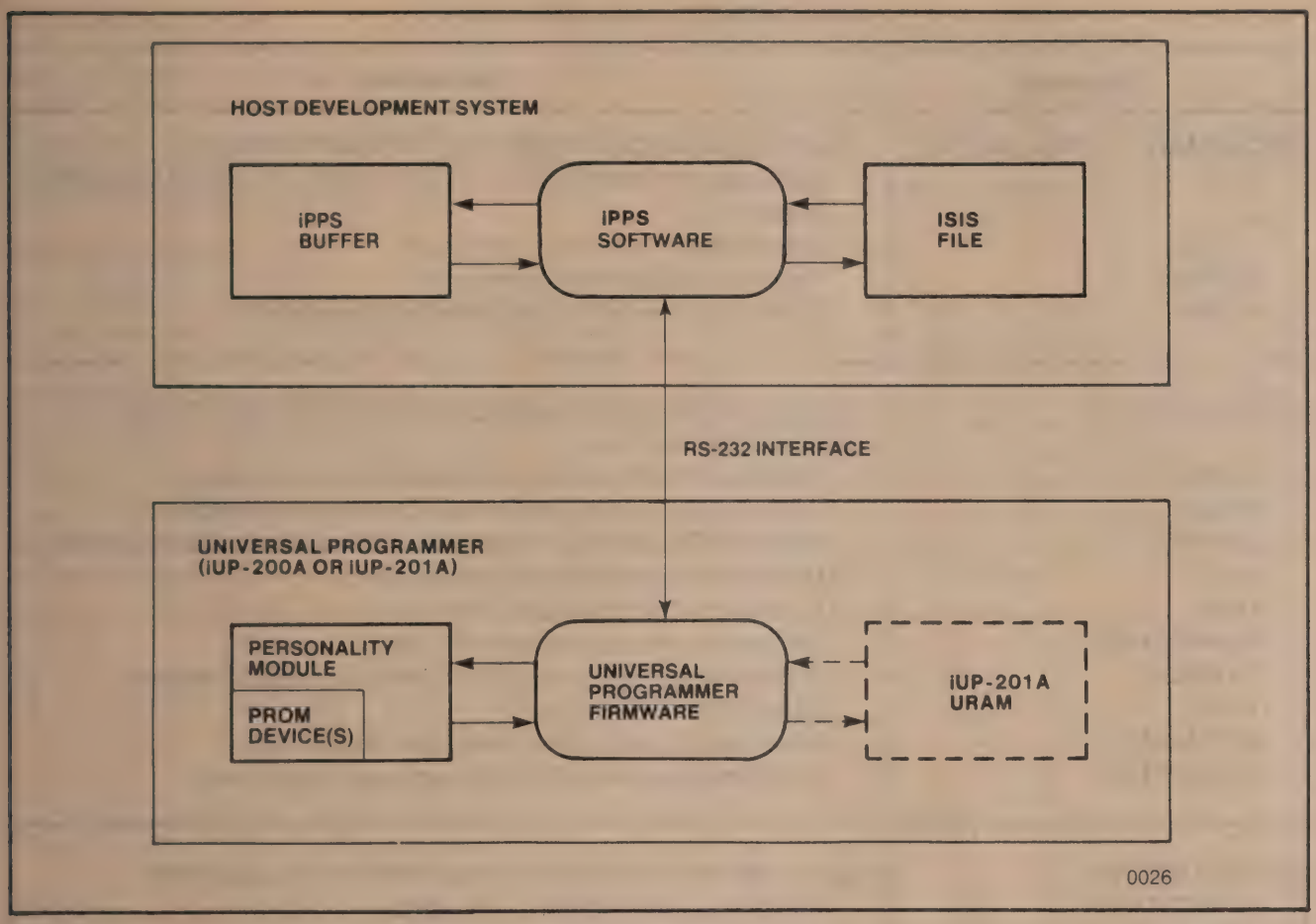


Figure 2 On-Line System Data Flow

- Verifies PROM data with buffer data
- Locks EPROM memory from unauthorized access (on devices which support this feature)
- Prints PROM contents on the network or development system printer
- Performs interactive formatting operations such as interleaving, nibble swapping, bit reversal, and block moves
- Programs multiple PROMs from the source file, prompting the user to insert new PROMs
- Uses a buffer to change PROM contents

All iPPS commands, as well as program address and data information, are entered through the development system ASCII keyboard and displayed on the system CRT. Table 1 summarizes the iPPS commands.

The iPPS software lets the user load programs into a PROM from Intellec system memory or directly from a disk file. Access to the disk lets the user create and manipulate data in a virtual buffer with an address range up to 16M. This large block of data can be formatted into different PROM word sizes for program storage into several different PROM types. In addition, a program stored in the target PROM, the Intellec system memory, or a system disk file can be interleaved with a second program and entered into a specific target PROM or PROMs.

The iPPS software supports data manipulation in the following Intel formats: 8080 hexadecimal ASCII, 8080 absolute object, 8086 hexadecimal ASCII, 8086 absolute object, and 80286 absolute object. Addresses and data can be displayed in binary, octal, decimal, or hexadecimal. The user can easily change default data formats as well as number bases.

The user invokes the iPPS software from the ISIS operating system (Intellec 800, Series II, and Series III, versions V3.4 and later; Series IV, versions V1.0 and later). The software can be run under control of ISIS submit files, thereby freeing the user from repetitious command entry.

Table 1 iPPS Command Summary

Command	Description
PROGRAM CONTROL GROUP	CONTROLS EXECUTION OF THE iPPS SOFTWARE.
EXIT	Exits the iPPS software and returns control to the ISIS operating system.
<ESC>	Terminates the current command.
REPEAT	Repeats the previous command.
ALTER	Edits and re-executes the previous command.
UTILITY GROUP	DISPLAYS USER INFORMATION AND STATUS AND SETS DEFAULT VALUES.
DISPLAY	Displays PROM, buffer, or file data on the console.
PRINT	Prints PROM, buffer, or file data on the local printer.
QUEUE	Prints PROM, buffer, or file data on the network spooled printer.
HELP	Displays user assistance information.
MAP	Displays buffer structure and status.
BLANKCHECK	Checks for unprogrammed PROMs.
OVERLAY	Checks whether non-blank PROMs can be programmed.
TYPE	Selects the PROM type.
INITIALIZE	Initializes the default number base and file type.
WORKFILES	Specifies the drive device for temporary work files.
BUFFER GROUP	EDITS, MODIFIES, AND VERIFIES DATA IN THE BUFFER.
SUBSTITUTE	Examines and modifies buffer data.
LOADDATA	Loads a section of the buffer with a constant.
VERIFY	Verifies data in the PROM with buffer data.
FORMATTING GROUP	REARRANGES DATA FROM THE PROM, BUFFER, OR FILE.
FORMAT	Formats and interleaves buffer, PROM, or file data.
COPY GROUP	COPIES DATA FROM ONE DEVICE TO ANOTHER.
COPY (file to PROM)	Programs the PROM with data in a file on disk.
COPY (PROM to file)	Saves PROM data in a file on disk.
COPY (buffer to PROM)	Programs the PROM with data from the buffer.
COPY (PROM to buffer)	Loads the buffer with data in the PROM.
COPY (buffer to file)	Saves the contents of the buffer in a file on disk.
COPY (file to buffer)	Loads the buffer from a file on disk.
COPY (file to URAM)	Loads file data into the iUP RAM (iUP-201A model only).
COPY (URAM to file)	Saves iUP URAM data in a file on disk (iUP-201A model only).
COPY (buffer to URAM)	Loads the buffer into the iUP URAM (iUP-201A model only).
COPY (URAM to buffer)	Loads iUP URAM data into the buffer (iUP-201A model only).
SECURITY GROUP	LOCKS SELECTED DEVICES TO PREVENT UNAUTHORIZED ACCESS.
KEYLOCK	Locks the PROM from unauthorized access.

System Expansion

The iUP-200A universal programmer can be easily upgraded (by the user) to an iUP-201A universal programmer for off-line operation. The upgrade kit (iUP-PAK-A) is available from Intel or your local Intel distributor.

Off-line System

The iUP-201A universal programmer has all the on-line features of the iUP-200A universal programmer plus off-line editing, PROM duplication, program verification, and locking of PROM memory independent of the host system. The iUP-201A universal programmer also accepts Intel hexadecimal programs developed on non-Intel development systems. Just a few keystrokes download the program into the iUP RAM for editing and loading into a PROM.

Off-line commands are entered using the off-line command keys summarized in Table 2.

In addition to the hardware components included as part of the iUP-200A, the iUP-201A contains a 24-character alphanumeric display, full hexadecimal 12-function keypad, and 32K bytes of iUP RAM. Figure 3 illustrates the iUP-201A keyboard and display.

The two logical devices accessible during off-line operation are the PROM device and the iUP RAM. A typical operation is copying the data from a PROM (or ROM) into the iUP RAM, modifying this data in iUP RAM, and programming the modified data back into a PROM device. The address range of the iUP RAM is automatically determined by the universal programmer when PROM type selection is made. Figure 4 shows the off-line system data flow.

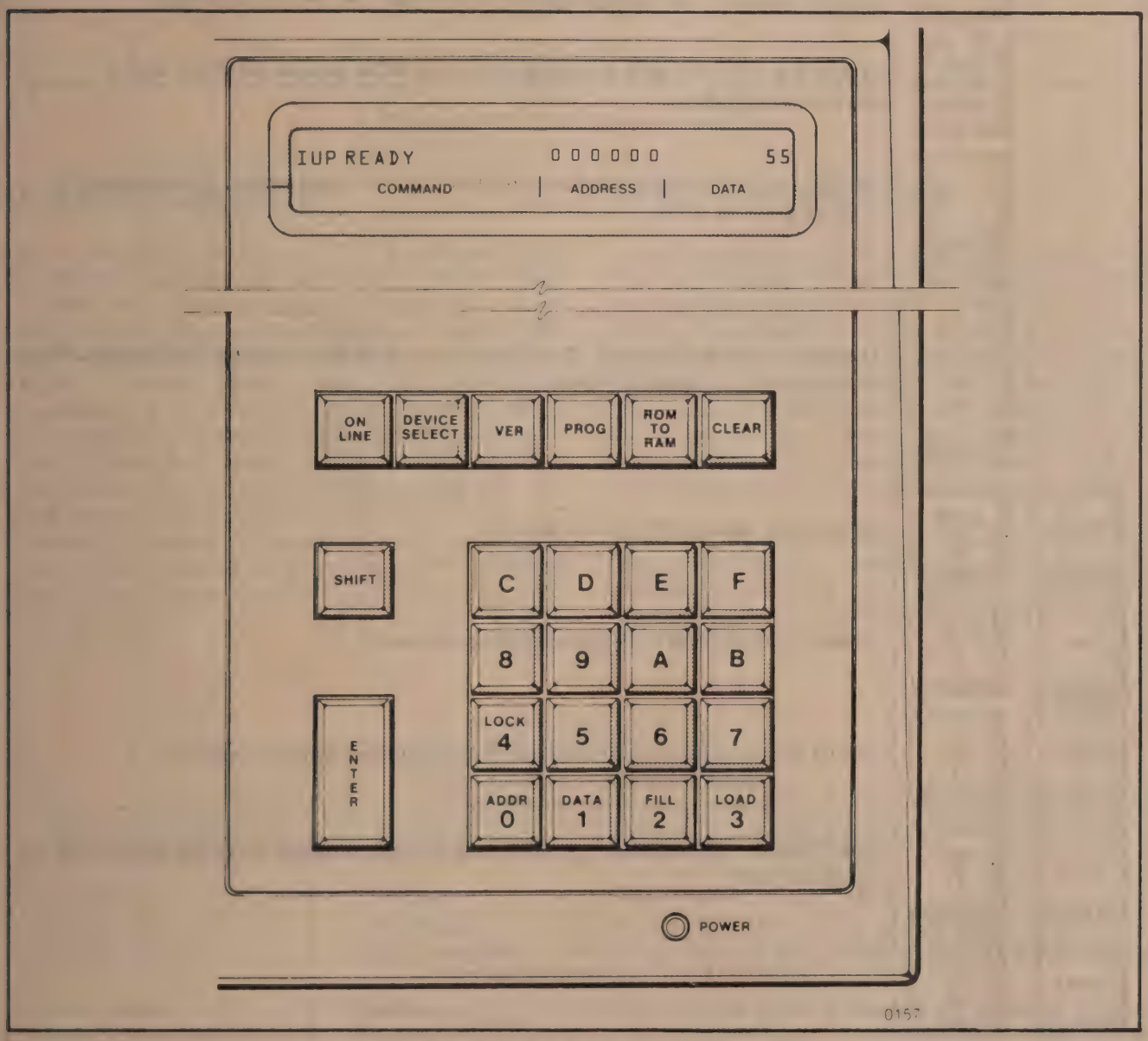


Figure 3 iUP-201A Keyboard and Display

Order Number: 210319-003

Table 2 Off-Line Command Keys Summary

Key	Function
ON LINE	Selects either on-line or off-line operation. When on-line, all other function keys are disabled.
DEVICE SELECT	Selects the PROM type when using a personality module able to program multiple PROM devices.
VER	Verifies the contents of the installed PROM device with the contents of the iUP RAM. The universal programmer display indicates the address and the XOR of any mismatches.
PROG	Performs a device blank check and then programs the target PROM with data from the iUP RAM. If the blank check fails, pressing PROG again performs an overlay check to verify that non-blank PROMs can be programmed.
ROM TO RAM	Loads the iUP RAM with the data from the PROM device installed in the personality module.
CLEAR	Terminates the current off-line function, clears a user entry, or restores the display after an error.
ENTER	Transfers information from the universal programmer display (addresses, data, or baud rate) into the iUP RAM.
SHIFT ADDR 0	Selects an address field for display.
SHIFT DATA 1	Selects a data field for keypad editing and entry.
SHIFT FILL 2	Loads a contiguous section of iUP RAM locations with a constant.
SHIFT LOAD 3	Downloads Intel hexadecimal data from any development system which has an RS-232C port.
SHIFT LOCK 4	Locks a PROM from unauthorized access.

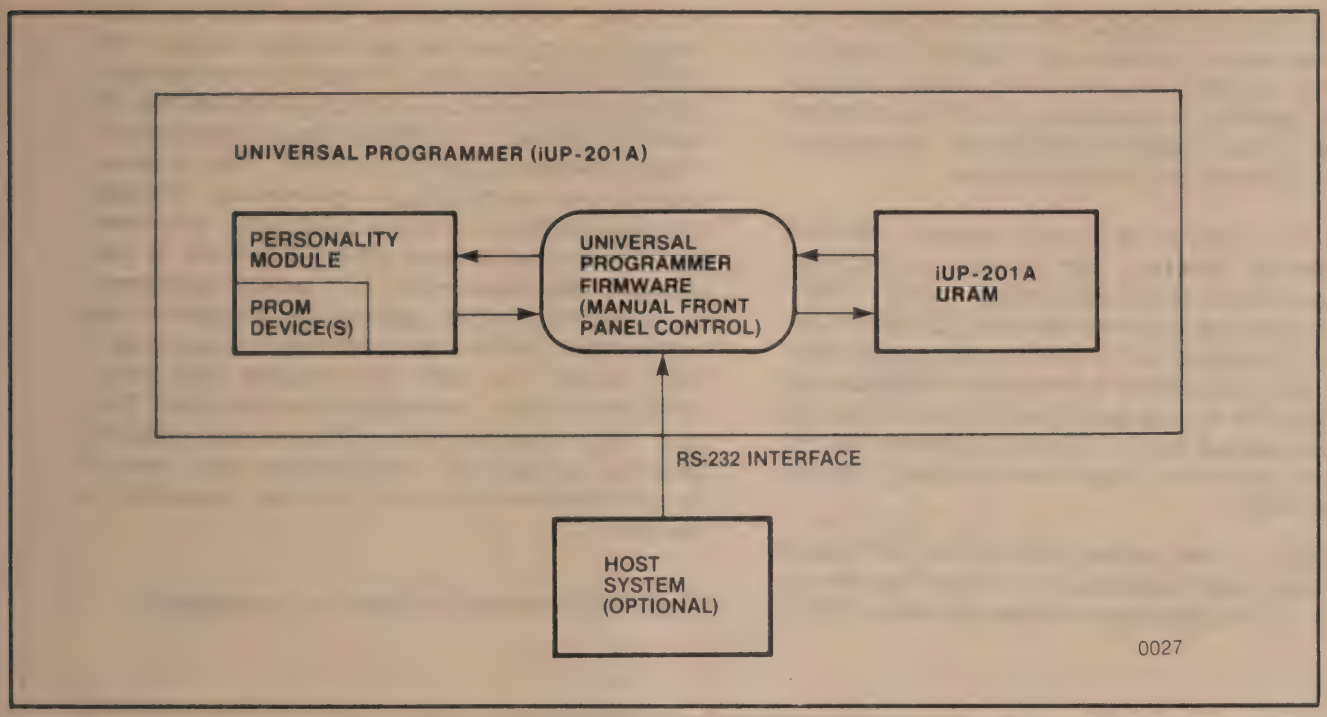


Figure 4 Off-Line System Data Flow

SYSTEM DIAGNOSTICS

Both the iUP-200A and iUP-201A universal programmers include self-contained system diagnostics that verify system operation and aid the user in fault isolation. Diagnostics are performed on the power supply, CPU internal firmware ROM, internal RAM, timer, the iUP-201A keyboard, and the iUP RAM. In addition, tests are made on any personality module installed in the programmer the first time the module is accessed. The personality module tests include the power select circuitry and up to 4K of module firmware. Straight-forward messages are provided on the development system display in on-line mode and on the iUP-201A display in off-line mode.

PERSONALITY MODULES

A personality module is the interface between the iUP-200A/iUP-201A universal programmer (or an iPDS system) and a selected PROM (or ROM). Personality modules contain all the hardware and firmware for reading and programming a family of Intel devices. Each personality module is a single molded unit inserted into the front panel of the universal programmer. No additional adapters or sockets are needed. Table 3 lists the available personality modules.

Each personality module connects to the universal programmer through a 41-pin connector. Module firmware is uploaded into the iUP RAM and executed by the internal 8085A processor.

Table 3 iUP Personality Modules

Personality Module	PROM Type	PROMs and ROMs Supported
iUP-Fast 27/K	EPROM	2764, 2764A, 27128, 27256
iUP-F27/128	E^2/EPROM	2716, 2732, 2732A, 2764, 27128, 2815, 2816
iUP-F87/51A	Microcontroller	8748, 8748H, 8048, 8749H, 8048H, 8049, 8049H, 8050H, 8751, 8751H, 8051
iUP-F87/44A	Peripheral	8741A, 8041A, 8742, 8042, 8744H, 8044AH, 8755A

The personality module firmware contains routines necessary to read and program a family of PROMs. In addition, the personality module sends specific information about the selected PROM to the universal programmer to help perform PROM device integrity checks.

LEDs on each personality module indicate operational status. On some personality modules a column of LEDs indicate which PROM device type the user has selected. On some personality modules an LED below the socket indicates which socket is to be used. A red indicator light tells the user when power is being supplied to the selected device. Figure 5 shows the personality modules supported on the universal programmer.

In addition to the testing done by the iUP system self-tests, each personality module contains diagnostic firmware that performs selected PROM tests and indicates status. These tests are performed in both on-line and off-line modes. The PROM installation test verifies that the device is installed in the module correctly and that the ZIF socket is closed. The PROM blank check determines whether a device is blank. The universal programmer automatically determines whether the blank state is all zeros or all ones. The overlay check (performed when a PROM is not blank) determines which bits are programmed, compares those bits against the program to be loaded, and allows programming to continue if they match. As with the system self-tests, straight-forward messages are provided. The user can invoke all of the PROM device integrity checks except the installation test (which occurs automatically any time an operation is selected).

Figure 6 illustrates a typical testing sequence.

Figure 5 Personality Modules

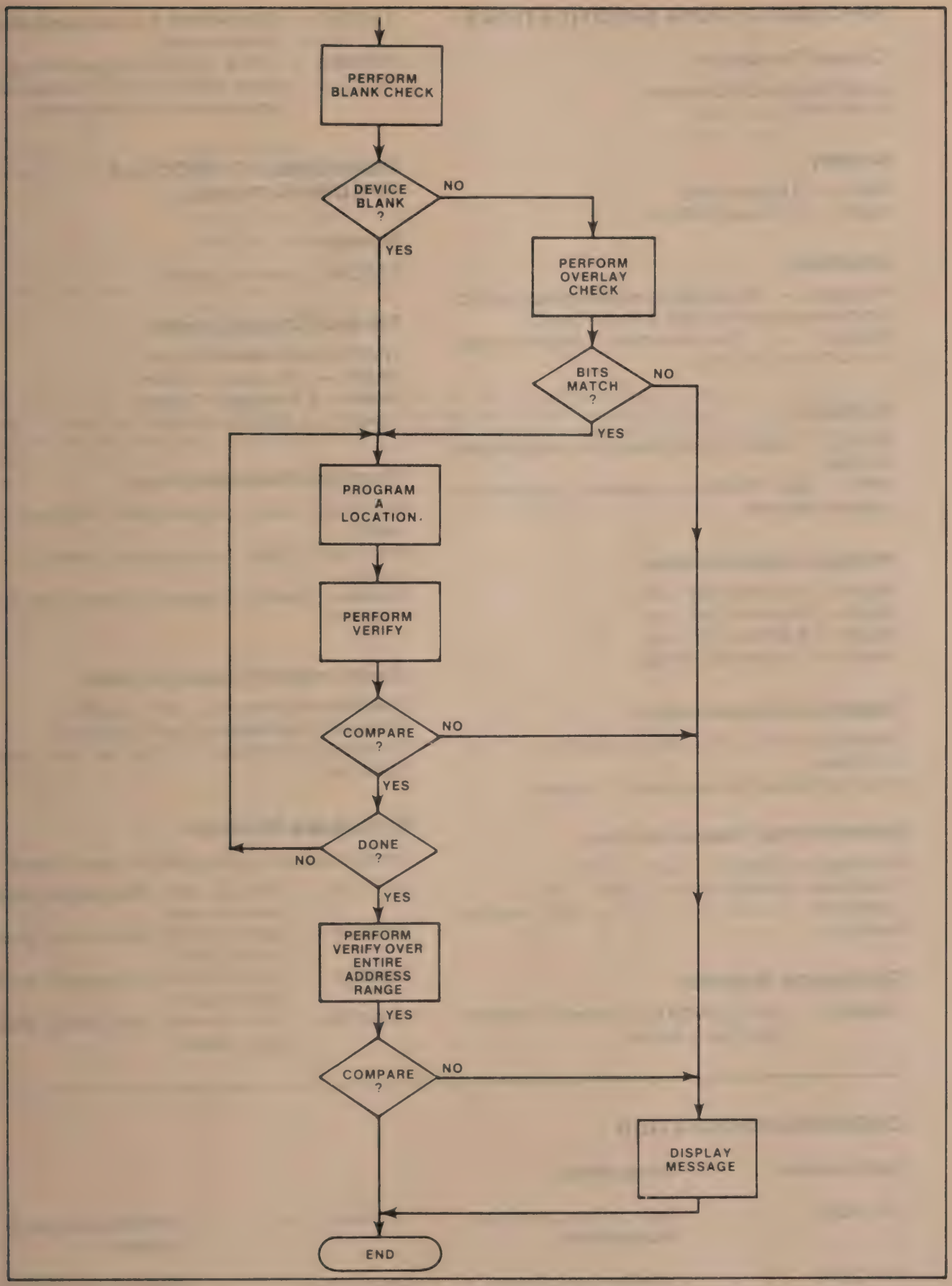


Figure 6 PROM Testing Sequence

Order Number: 210319-003

iUP-200A/iUP-201A SPECIFICATIONS

Control Processor

Intel 8085A microprocessor
6.144 MHz clock rate

Memory

RAM — 4.3 bytes static
ROM — 12K bytes EPROM

Interfaces

Keyboard — 16-character hexadecimal and 12-function keypad (iUP-201A model only)
Display — 24-character alphanumeric (iUP-201A model only)

Software

Monitor — system controller in pre-programmed EPROM
iPPS — Intel PROM programming software on supplied diskette

Physical Characteristics

Depth — 15 inches (38.1 cm)
Width — 15 inches (38.1 cm)
Height — 6 inches (15.2 cm)
Weight — 15 pounds (6.9 kg)

Electrical Characteristics

Selectable 100, 120, 200, or 240 Vac ± 10%; 50-60 Hz
Maximum power consumption — 80 watts

Environmental Characteristics

Reading temperature — 10°C to 40°C
Programming temperature — 25°C ± 5°
Operating humidity — 10% to 85% relative humidity

Reference Material

164852 — *iUP-200A/201A Universal Programmer User's Guide.*
164861 — *iPPS PROM Programming Software User's Guide.*
164853 — *iPPS PROM Programming Software/iUP-200A/201A Universal Programmer Pocket Reference.*

PERSONALITY MODULE SPECIFICATIONS

Memory

EPROM — up to 4K bytes

Physical Characteristics

Width — 5.5 inches (1.4 cm)
Height — 1.6 inches (4.1 cm)
Depth — 7.0 inches (17.8 cm)
Weight — 1 pound (.45 kg)

Electrical Characteristics

Maximum power consumption (module) — 7.5 watts
Maximum power consumption (device) — 2.5 watts
Maximum power consumption (total from iUP) — 10 watts

Environmental Characteristics

Reading temperature — 10°C to 40°C
Programming temperature — 25°C ± 5°
Operating humidity — 10% to 85% relative humidity

Reference Material

Appropriate personality module user's guide:

164376 — *iUP-Fast 27/K Personality Module User's Guide.*
162848 — *IUP-F27/128 Personality Module User's Guide.*
164855 — *iUP-F87/51A Personality Module User's Guide.*
164853 — *iUP-F87/44A Personality Module User's Guide.*

ORDERING INFORMATION

Part number	Description		
iUP-200A	Intel on-line universal programmer	iUP-Fast 27/K*	EPROM personality module
iUP-201A	Intel on-line/off-line universal programmer	iUP-F27/128	EPROM and E²PROM personality module

Order Number: 210319-003

iUP-F87/51A	Microcontroller personality module
iUP-F87/44A	Peripheral personality module
iUP-200/201 U1 Upgrade Kit	Upgrades an iUP-200/201 universal programmer to an iUP-200A/201A universal programmer
iUP-PAK-A Upgrade Kit	Upgrades an iUP-200A universal programmer to an iUP-201A universal programmer

*The iUP-Fast 27/K personality module can be used only with an iUP-200A/201A universal programmer or an iUP-200 /iUP-201 universal programmer upgraded to an A with the iUP-200/201 U1 upgrade kit. If used in an iPDS, this personality module requires version 1.4 or later of the iPPS-iPDS software. All iPDS-140 units shipped after June 1984 will contain this software.

Order Number: 210319-003

Video Display

7

intel®

8275H
PROGRAMMABLE CRT CONTROLLER

- **Programmable Screen and Character Format**
- **6 Independent Visual Field Attributes**
- **11 Visual Character Attributes (Graphic Capability)**
- **Cursor Control (4 Types)**
- **Light Pen Detection and Registers**
- **MCS-51®, MCS-85®, iAPX 86, and iAPX 88 Compatible**
- **Dual Row Buffers**
- **Programmable DMA Burst Mode**
- **Single +5V Supply**
- **High Performance HMOS-II**

The Intel® 8275H Programmable CRT Controller is a single chip device to interface CRT raster scan displays with Intel® microcomputer systems. It is manufactured on Intel's advanced HMOS-II process. Its primary function is to refresh the display by buffering the information from main memory and keeping track of the display position of the screen. The flexibility designed in the 8275H will allow simple interface to almost any raster scan CRT display with a minimum of external hardware and software overhead.

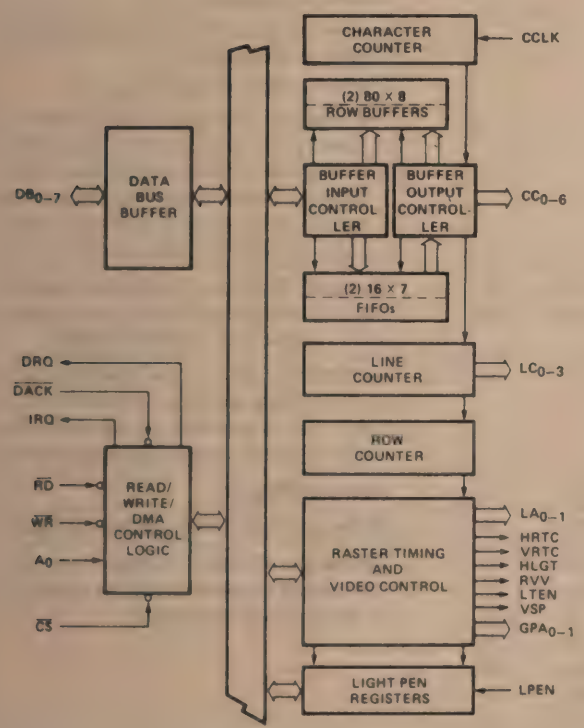


Figure 1. Block Diagram

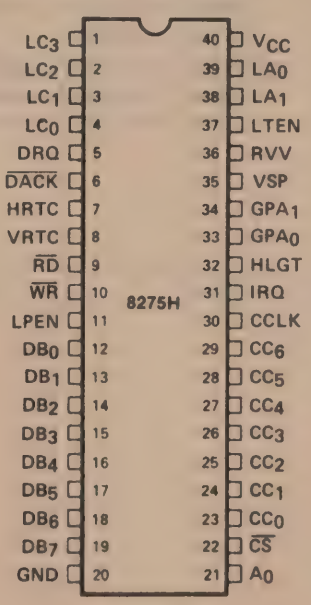


Figure 2. Pin Configuration

Intel Corporation Assumes No Responsibility for the Use of Any Circuitry Other Than Circuitry Embodied in an Intel Product. No Other Circuit Patent Licenses are Implied.
©INTEL CORPORATION, 1984

SEPTEMBER 1984
210464-002

Table 1. Pin Descriptions

Symbol	Pin No.	Type	Name and Function
LC_3 LC_2 LC_1 LC_0	1 2 3 4	O	**Line Count:** Output from the line counter which is used to address the character generator for the line positions on the screen.
DRQ	5	O	**DMA Request:** Output signal to the 8257 DMA controller requesting a DMA cycle.
$\overline{DACK}$	6	I	**DMA Acknowledge:** Input signal from the 8257H DMA controller acknowledging that the requested DMA cycle has been granted.
HRTC	7	O	**Horizontal Retrace:** Output signal which is active during the programmed horizontal retrace interval. During this period the VSP output is high and the LTEN output is low.
VRTC	8	O	**Vertical Retrace:** Output signal which is active during the programmed vertical retrace interval. During this period the VSP output is high and the LTEN output is low.
$\overline{RD}$	9	I	**Read Input:** A control signal to read registers.
$\overline{WR}$	10	I	**Write Input:** A control signal to write commands into the control registers or write data into the row buffers during a DMA cycle.
LPEN	11	I	**Light Pen:** Input signal from the CRT system signifying that a light pen signal has been detected.
DB_0 DB_1 DB_2 DB_3 DB_4 DB_5 DB_6 DB_7	12 13 14 15 16 17 18 19	I/O	**Bi-Directional Three-State Data Bus Lines:** The outputs are enabled during a read of the C or P ports.
Ground	20		**Ground.**
V_{CC}	40		**+5V Power Supply.**
LA_0 LA_1	39 38	O	**Line Attribute Codes:** These attribute codes have to be decoded externally by the dot/timing logic to generate the horizontal and vertical line combinations for the graphic displays specified by the character attribute codes.
LTEN	37	O	**Light Enable:** Output signal used to enable the video signal to the CRT. This output is active at the programmed underline cursor position, and at positions specified by attribute codes.
RVV	36	O	**Reverse Video:** Output signal used to indicate the CRT circuitry to reverse the video signal. This output is active at the cursor position if a reverse video block cursor is programmed or at the positions specified by the field attribute codes.
VSP	35	O	**Video Suppression:** Output signal used to blank the video signal to the CRT. This output is active: —during the horizontal and vertical retrace intervals. —at the top and bottom lines of rows if underline is programmed to be number 8 or greater. —when an end of row or end of screen code is detected. —when a DMA underrun occurs. —at regular intervals (1/16 frame frequency for cursor, 1/32 frame frequency for character and field attributes)—to create blinking displays as specified by cursor, character attribute, or field attribute programming.
GPA_1 GPA_0	34 33	O	**General Purpose Attribute Codes:** Outputs which are enabled by the general purpose field attribute codes.
HLGT	32	O	**Highlight:** Output signal used to intensify the display at particular positions on the screen as specified by the character attribute codes or field attribute codes.
IRQ	31	O	**Interrupt Request.**
CCLK	30	I	**Character Clock (from dot/timing logic).**
CC_6 CC_5 CC_4 CC_3 CC_2 CC_1 CC_0	29 28 27 26 25 24 23	O	**Character Codes:** Output from the row buffers used for character selection in the character generator.
CS	22	I	**Chip Select:** The read and write are enabled by CS.
A_0	21	I	**Port Address:** A high input on A_0 selects the "C" port or command registers and a low input selects the "P" port or parameter registers.

FUNCTIONAL DESCRIPTION

Data Bus Buffer

This 3-state, bidirectional, 8-bit buffer is used to interface the 8275H to the system Data Bus.

This functional block accepts inputs from the System Control Bus and generates control signals for overall device operation. It contains the Command, Parameter, and Status Registers that store the various control formats for the device functional definition.

A_0	OPERATION	REGISTER
0	Read	PREG
0	Write	PREG
1	Read	SREG
1	Write	CREG

A_0	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	
0	1	0	0	Write 8275H Parameter
0	0	1	0	Read 8275H Parameter
1	1	0	0	Write 8275H Command
1	0	1	0	Read 8275H Status
X	1	1	0	Three-State
X	X	X	1	Three-State

$\overline{RD}$ (Read)

A "low" on this input informs the 8275H that the CPU is reading data or status information from the 8275H.

$\overline{WR}$ (Write)

A "low" on this input informs the 8275H that the CPU is writing data or control words to the 8275H.

$\overline{CS}$ (Chip Select)

A "low" on this input selects the 8275H. No reading or writing will occur unless the device is selected. When $\overline{CS}$ is high, the Data Bus in the float state and $\overline{RD}$ and $\overline{WR}$ will have no effect on the chip.

DRQ (DMA Request)

A "high" on this output informs the DMA Controller that the 8275H desires a DMA transfer.

$\overline{DACK}$ (DMA Acknowledge)

A "low" on this input informs the 8275H that a DMA cycle is in progress.

IRQ (Interrupt Request)

A "high" on this output informs the CPU that the 8275H desires interrupt service.

FUNCTIONAL DESCRIPTION

Character Counter

The Character Counter is a programmable counter that is used to determine the number of characters to be displayed per row and the length of the horizontal retrace interval. It is driven by the CCLK (Character Clock) input, which should be a derivative of the external dot clock.

Line Counter

The Line Counter is a programmable counter that is used to determine the number of horizontal lines (Sweeps) per character row. Its outputs are used to address the external character generator ROM.

Row Counter

The Row Counter is a programmable counter that is used to determine the number of character rows to be displayed per frame and length of the vertical retrace interval.

Light Pen Registers

The Light Pen Registers are two registers that store the contents of the character counter and the row counter whenever there is a rising edge on the LPEN (Light Pen) input.

Note: Software correction is required.

Raster Timing and Video Controls

The Raster Timing circuitry controls the timing of the HRTC (Horizontal Retrace) and VRTC (Vertical Retrace) outputs. The Video Control circuitry controls the generation of LA_{0-1} (Line Attribute), HGLT (Highlight), RVV (Reverse Video), LTEN (Light Enable), VSP (Video Suppress), and GPA_{0-1} (General Purpose Attribute) outputs.

Row Buffers

The Row Buffers are two 80 character buffers. They are filled from the microcomputer system memory with the character codes to be displayed. While one row buffer is displaying a row of characters, the other is being filled with the next row of characters.

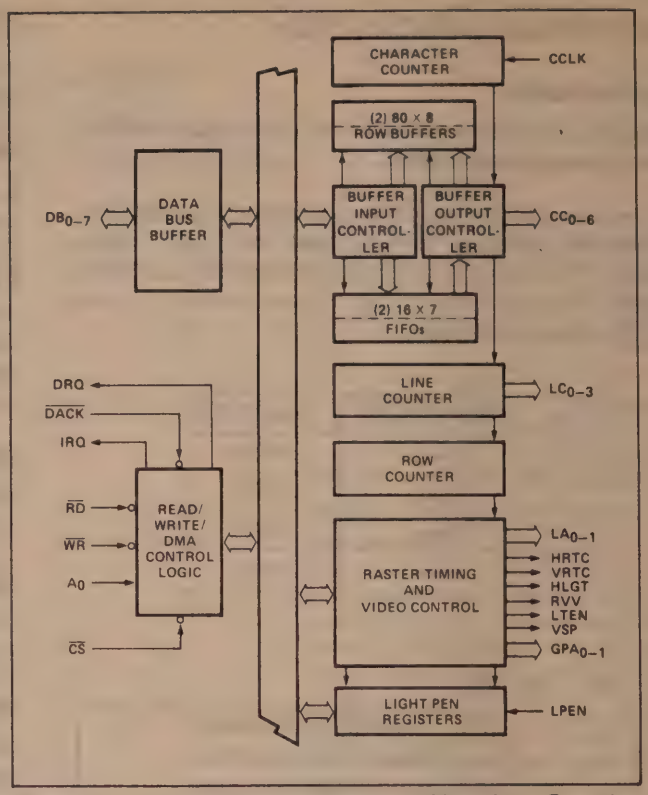


Figure 3. 8275H Block Diagram Showing Counter and Register Functions

FIFOs

There are two 16 character FIFOs in the 8275H. They are used to provide extra row buffer length in the Transparent Attribute Mode (see Detailed Operation section).

Buffer Input/Output Controllers

The Buffer Input/Output Controllers decode the characters being placed in the row buffers. If the character is a character attribute, field attribute or special code, these controllers control the appropriate action. (Examples: An "End of Screen–Stop DMA" special code will cause the Buffer Input Controller to stop further DMA requests. A "Highlight" field attribute will cause the Buffer Output Controller to activate the HGLT output.)

SYSTEM OPERATION

The 8275H is programmable to a large number of different display formats. It provides raster timing, display row buffering, visual attribute decoding, cursor timing, and light pen detection.

It is designed to interface with the 8257 DMA Controller and standard character generator ROMs for dot matrix decoding. Dot level timing must be provided by external circuitry.

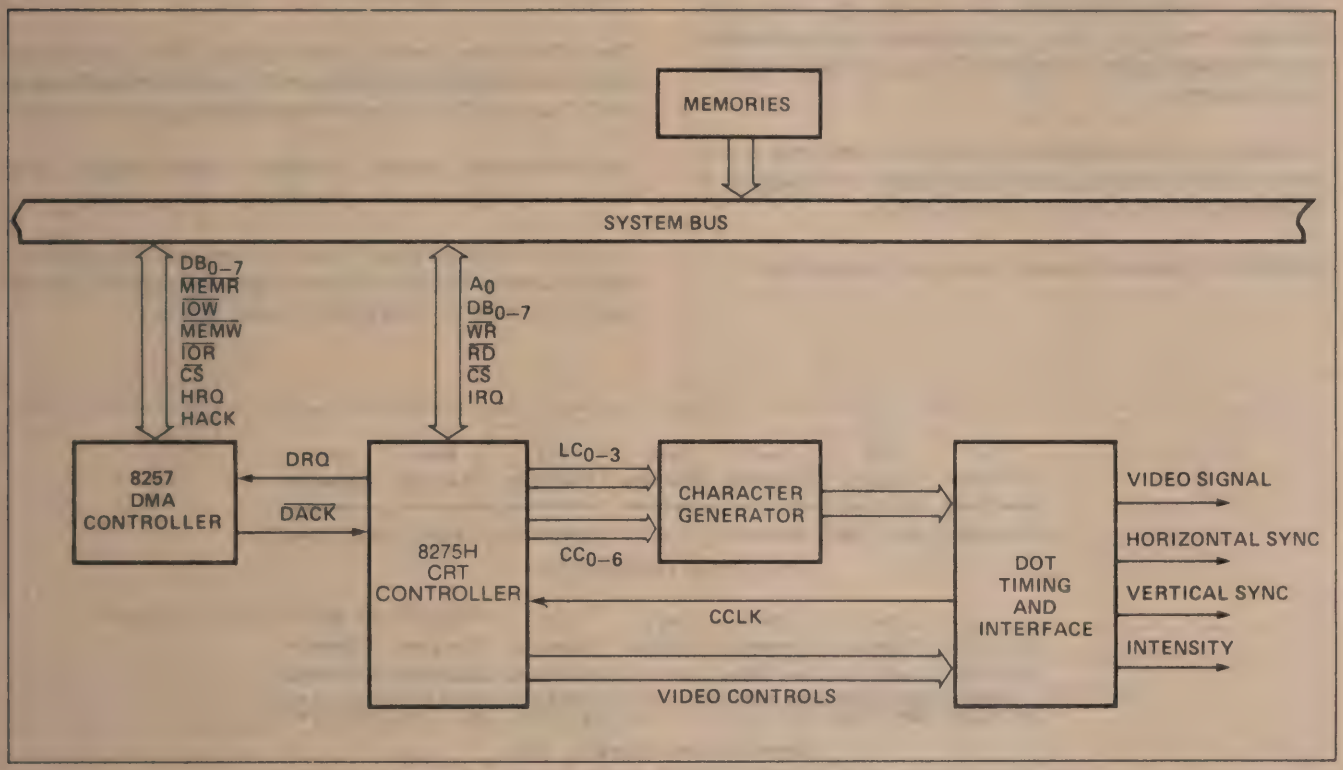


Figure 4. 8275H Systems Block Diagram Showing Systems Operation

General Systems Operational Description

The 8275H provides a "window" into the microcomputer system memory.

Display characters are retrieved from memory and displayed on a row by row basis. The 8275H has two row buffers. While one row buffer is being used for display, the other is being filled with the next row of characters to be displayed. The number of display characters per row and the number of character rows per frame are software programmable, providing easy interface to most CRT displays. (See Programming Section.)

The 8275H requests DMA to fill the row buffer that is not being used for display. DMA burst length and spacing is programmable. (See Programming Section.)

The 8275H displays character rows one line at a time.

The number of lines per character row, the underline position, and blanking of top and bottom lines are programmable. (See Programming Section.)

The 8275H provides special Control Codes which can be used to minimize DMA or software overhead. It also provides Visual Attribute Codes to cause special action or symbols on the screen without the use of the character generator (See Visual Attributes Section).

The 8275H also controls raster timing. This is done by generating Horizontal Retrace (HRTC) and Vertical Retrace (VRTC) signals. The timing of these signals is programmable.

The 8275H can generate a cursor. Cursor location and format are programmable. (See Programming Section.)

The 8275H has a light pen input and registers. The light pen input is used to load the registers. Light pen registers can be read on command. (See Programming Section.)

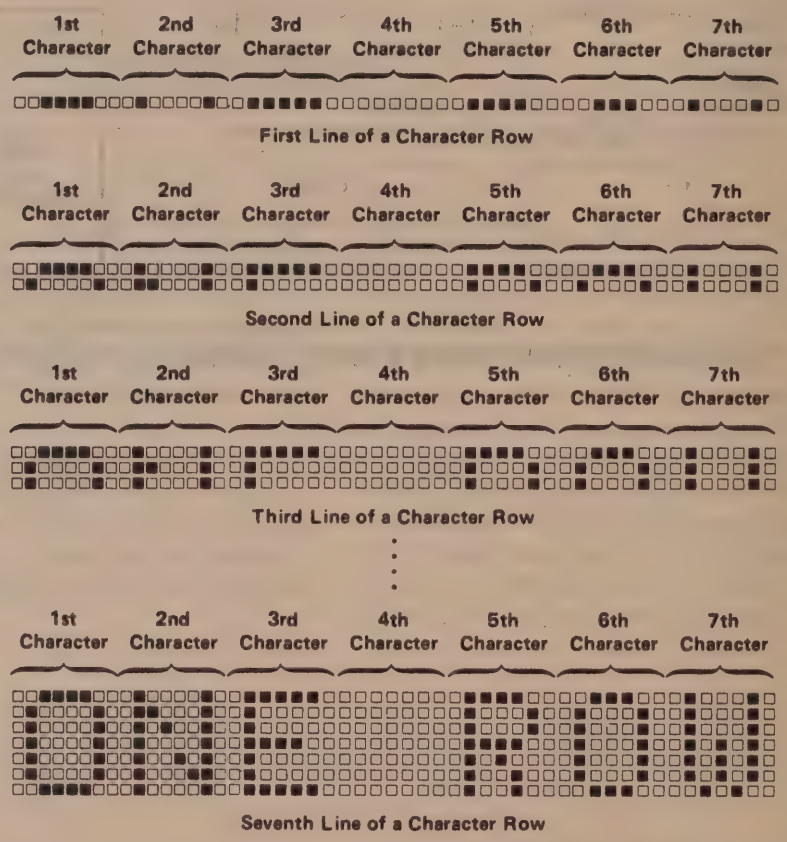


Figure 5. Display of a Character Row

Display Row Buffering

Before the start of a frame, the 8275H requests DMA and one row buffer is filled with characters.

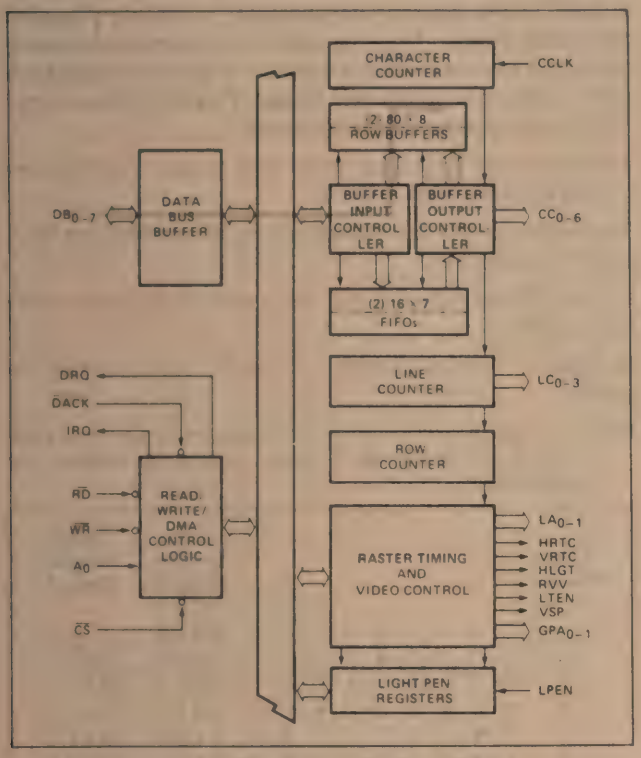


Figure 6. First Row Buffer Filled

When the first horizontal sweep is started, character codes are output to the character generator from the row buffer just filled. Simultaneously, DMA begins filling the other row buffer with the next row of characters.

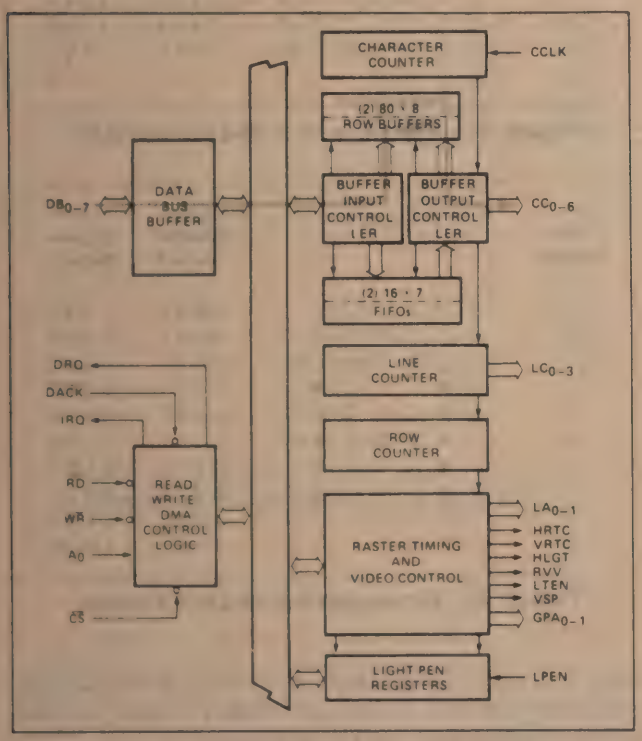


Figure 7. Second Buffer Filled, First Row Displayed

After all the lines of the character row are scanned, the roles of the two row buffers are reversed and the same procedure is followed for the next row.

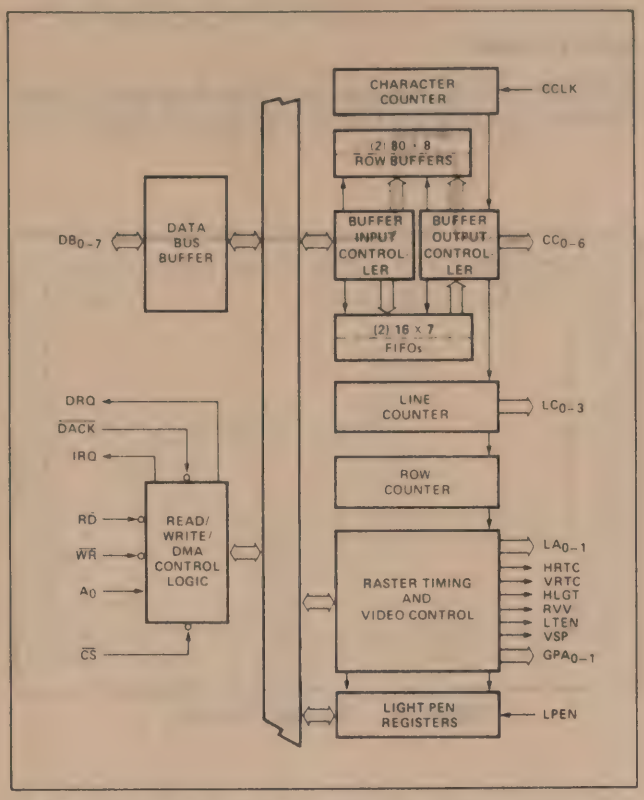


Figure 8. First Buffer Filled with Third Row, Second Row Displayed

This is repeated until all of the character rows are displayed.

Display Format

Screen Format

The 8275H can be programmed to generate from 1 to 80 characters per row, and from 1 to 64 rows per frame.

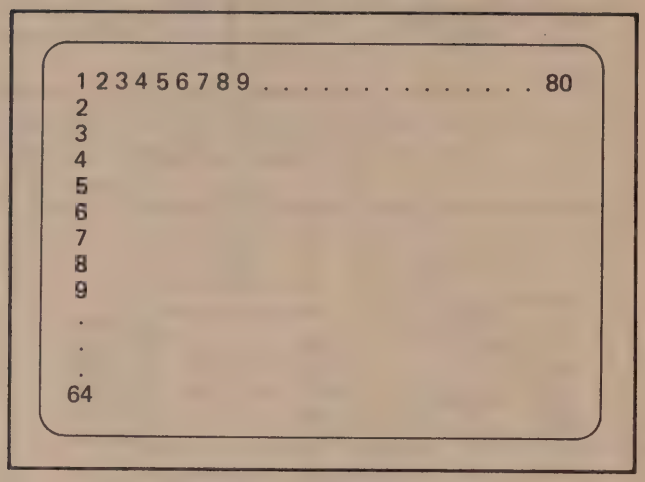


Figure 9. Screen Format

The 8275H can also be programmed to blank alternate rows. In this mode the first row is displayed, the second blanked, the third displayed, etc. DMA is not requested for the blanked rows.

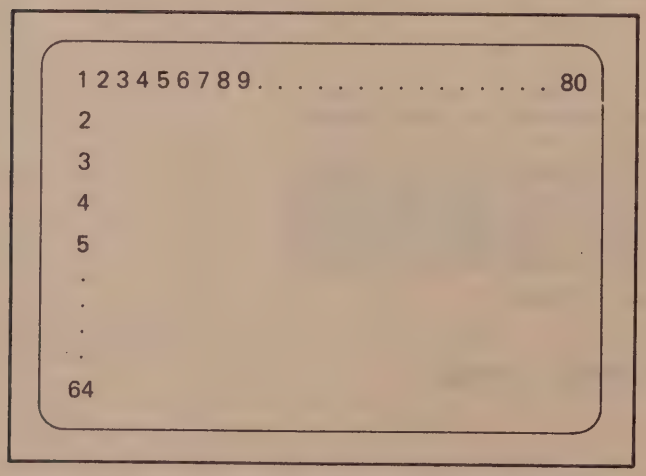


Figure 10. Blank Alternate Rows Mode

Row Format

The 8275H is designed to hold the line count stable while outputting the appropriate character codes during each horizontal sweep. The line count is incremented during horizontal retrace and the whole row of character codes are output again during the next sweep. This is continued until the whole character row is displayed.

The number of lines (horizontal sweeps) per character row is programmable from 1 to 16.

The output of the line counter can be programmed to be in one of two modes.

In mode 0, the output of the line *counter* is the same as the line *number*.

In mode 1, the line *counter* is offset by one from the line *number*.

Note: In mode 1, while the *first* line (line number 0) is being displayed, the *last* count is output by the line counter (see examples).

Line Number										Line Counter Mode 0	Line Counter Mode 1
0	□	□	□	□	□	□	□	□	□	0000	1111
1	□	□	□	□	■	□	□	□	□	0001	0000
2	□	□	□	■	□	■	□	□	□	0010	0001
3	□	□	■	□	□	□	■	□	□	0011	0010
4	□	■	□	□	□	□	□	■	□	0100	0011
5	□	■	□	□	□	□	□	■	□	0101	0100
6	□	■	■	■	■	■	■	■	□	0110	0101
7	□	■	□	□	□	□	□	■	□	0111	0110
8	□	■	□	□	□	□	□	■	□	1000	0111
9	□	■	□	□	□	□	□	■	□	1001	1000
10	□	□	□	□	□	□	□	□	□	1010	1001
11	□	□	□	□	□	□	□	□	□	1011	1010
12	□	□	□	□	□	□	□	□	□	1100	1011
13	□	□	□	□	□	□	□	□	□	1101	1100
14	□	□	□	□	□	□	□	□	□	1110	1101
15	□	□	□	□	□	□	□	□	□	1111	1110

Figure 11. Example of a 16-Line Format

Line Number								Line Counter Mode 0	Line Counter Mode 1
0	□	□	□	□	□	□	□	0000	1001
1	□	□	□	■	□	□	□	0001	0000
2	□	□	■	□	■	□	□	0010	0001
3	□	■	□	□	□	■	□	0011	0010
4	□	■	□	□	□	■	□	0100	0011
5	□	■	■	■	■	■	□	0101	0100
6	□	■	□	□	□	■	□	0110	0101
7	□	■	□	□	□	■	□	0111	0110
8	□	□	□	□	□	□	□	1000	0111
9	□	□	□	□	□	□	□	1001	1000

Figure 12. Example of a 10-Line Format

Mode 0 is useful for character generators that leave address zero blank and start at address 1. Mode 1 is useful for character generators which start at address zero.

Underline placement is also programmable (from line *number* 0 to 15). This is independent of the line *counter* mode.

If the line *number* of the underline is greater than 7 (line *number* MSB = 1), then the top and bottom lines will be blanked.

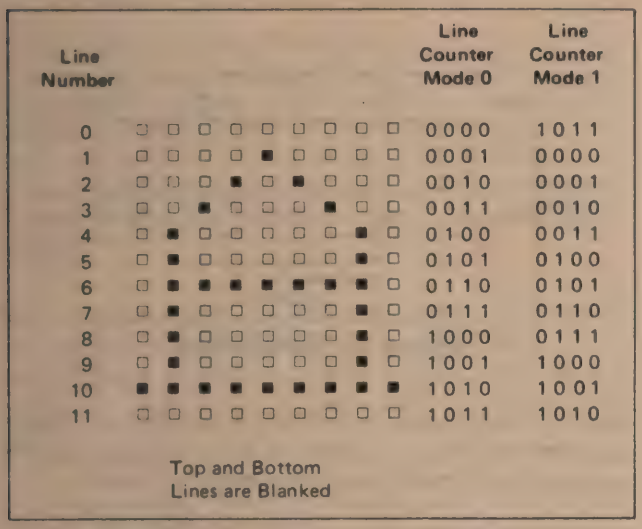


Figure 13. Underline in Line Number 10

If the line *number* of the underline is less than or equal to 7 (line *number* MSB = 0), then the top and bottom lines will *not* be blanked.

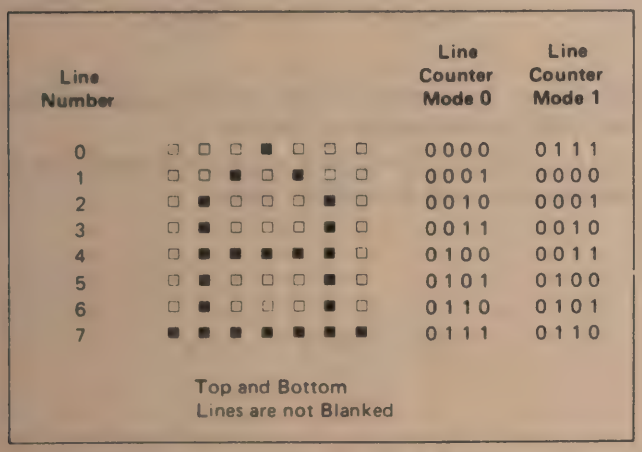


Figure 14. Underline in Line Number 7

If the line *number* of the underline is greater than the maximum number of lines, the underline will not appear.

Blanking is accomplished by the VSP (Video Suppression) signal. Underline is accomplished by the LTEN (Light Enable) signal.

Dot Format

Dot width and character width are dependent upon the external timing and control circuitry.

Dot level timing circuitry should be designed to accept the parallel output of the character generator and shift it out serially at the rate required by the CRT display.

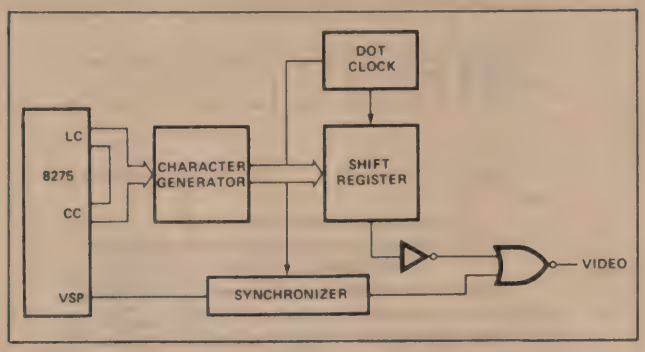


Figure 15. Typical Dot Level Block Diagram

Dot width is a function of dot clock frequency.

Character width is a function of the character generator width.

Horizontal character spacing is a function of the shift register length.

Note: Video control and timing signals must be synchronized with the video signal due to the character generator access delay.

Raster Timing

The character counter is driven by the character clock input (CCLK). It counts out the characters being displayed (programmable from 1 to 80). It then causes the line counter to increment, and it starts counting out the horizontal retrace interval (programmable from 2 to 32). This is constantly repeated.

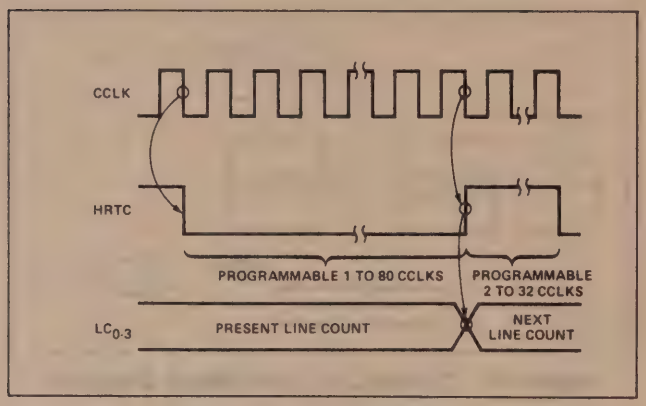


Figure 16. Line Timing

The line counter is driven by the character counter. It is used to generate the line address outputs (LC_{0-3}) for the character generator. After it counts all of the lines in a character row (programmable from 1 to 16), it increments the row counter, and starts over again. (See Character Format Section for detailed description of Line Counter functions.)

The row counter is an internal counter driven by the line counter. It controls the functions of the row buffers and counts the number of character rows displayed.

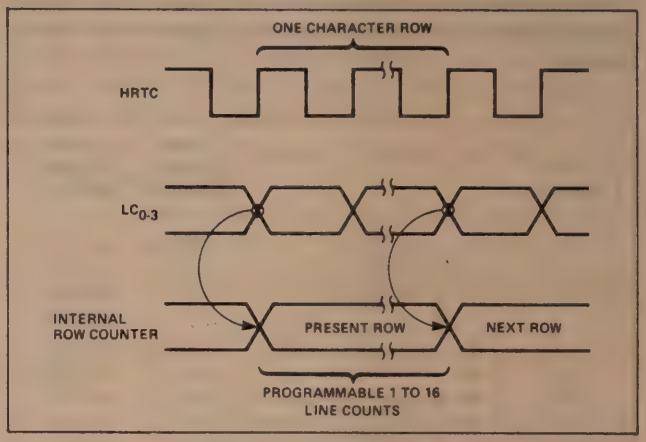


Figure 17. Row Timing

After the row counter counts all of the rows in a frame (programmable from 1 to 64), it starts counting out the vertical retrace interval (programmable from 1 to 4).

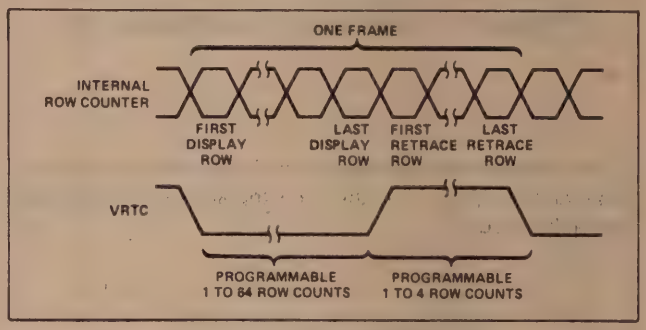


Figure 18. Frame Timing

The Video Suppression Output (VSP) is active during horizontal and vertical retrace intervals.

Dot level timing circuitry must synchronize these outputs with the video signal to the CRT Display.

DMA Timing

The 8275H can be programmed to request burst DMA transfers of 1 to 8 characters. The interval between bursts is also programmable (from 0 to 55 character clock periods ±1). This allows the user to tailor his DMA overhead to fit his system needs.

The first DMA request of the frame occurs one *row time* before the end of vertical retrace. DMA requests continue as programmed, until the row buffer is filled. If the row buffer is filled in the middle of a burst, the 8275H terminates the burst and resets the burst counter. No more DMA requests will occur until the *beginning* of the *next* row. At that time, DMA requests are activated as programmed until the other buffer is filled.

The first DMA request for a row will start at the first character clock of the preceding row. If the burst mode is used, the first DMA request may occur a number of character clocks later. This number is equal to the programmed burst space.

If, for any reason, there is a DMA underrun, a flag in the status word will be set.

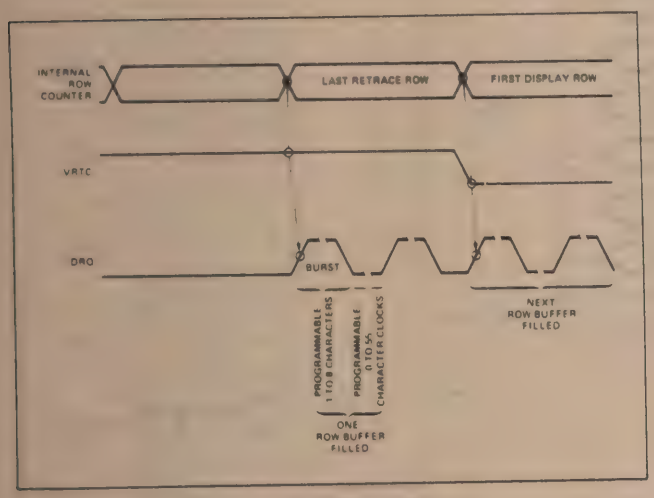


Figure 19. DMA Timing

The DMA controller is typically initialized for the next frame at the end of the current frame.

Interrupt Timing

The 8275H can be programmed to generate an interrupt request at the end of each frame. This can be used to reinitialize the DMA controller. If the 8275H interrupt enable flag is set, an interrupt request will occur at the *beginning* of the *last display row.*

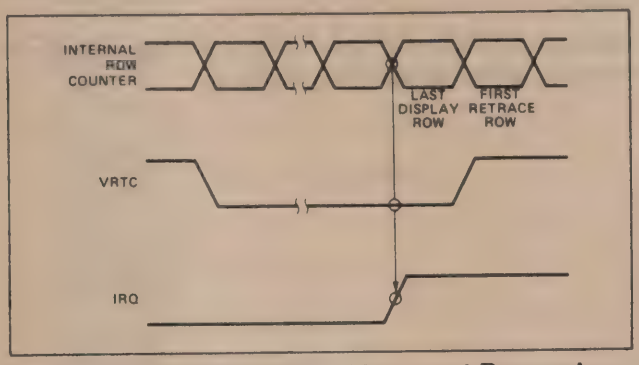


Figure 20. Beginning of Interrupt Request

IRQ will go inactive after the status register is read.

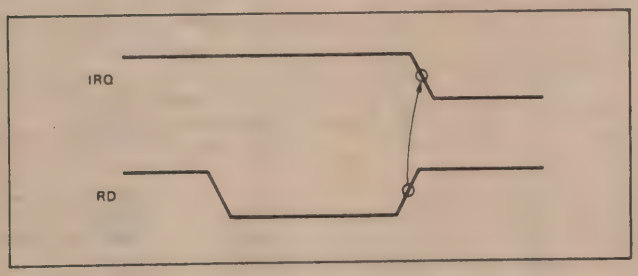


Figure 21. End of Interrupt Request

A reset command will also cause IRQ to go inactive, but this is not recommended during normal service.

Another method of reinitializing the DMA controller is to have the DMA controller itself interrupt on terminal count. With this method, the 8275H interrupt enable flag should not be set.

Note: Upon power-up, the 8275H Interrupt Enable Flag may be set. As a result, the user's cold start routine should write a reset command to the 8275H before system interrupts are enabled.

VISUAL ATTRIBUTES AND SPECIAL CODES

The characters processed by the 8275H are 8-bit quantities. The character code outputs provide the character generator with 7 bits of address. The Most Significant Bit is the extra bit and it is used to determine if it is a normal display character (MSB = 0), or if it is a Visual Attribute or Special Code (MSB = 1).

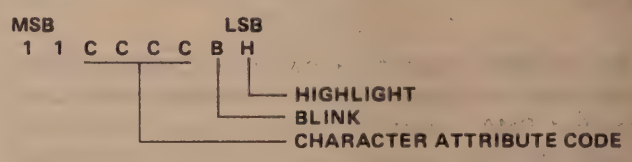
There are two types of Visual Attribute Codes. They are Character Attributes and Field Attributes.

Character Attribute Codes

Character attribute codes are codes that can be used to generate graphics symbols without the use of a character generator. This is accomplished by selectively activating the Line Attribute outputs (LA_{0-1}), the Video Suppression output (VSP), and the Light Enable output. The dot level timing circuitry can use these signals to generate the proper symbols.

Character attributes can be programmed to blink or be highlighted individually. Blinking is accomplished with the Video Suppression output (VSP). Blink frequency is equal to the screen refresh frequency divided by 32. Highlighting is accomplished by activating the Highlight output (HGLT).

Character Attributes



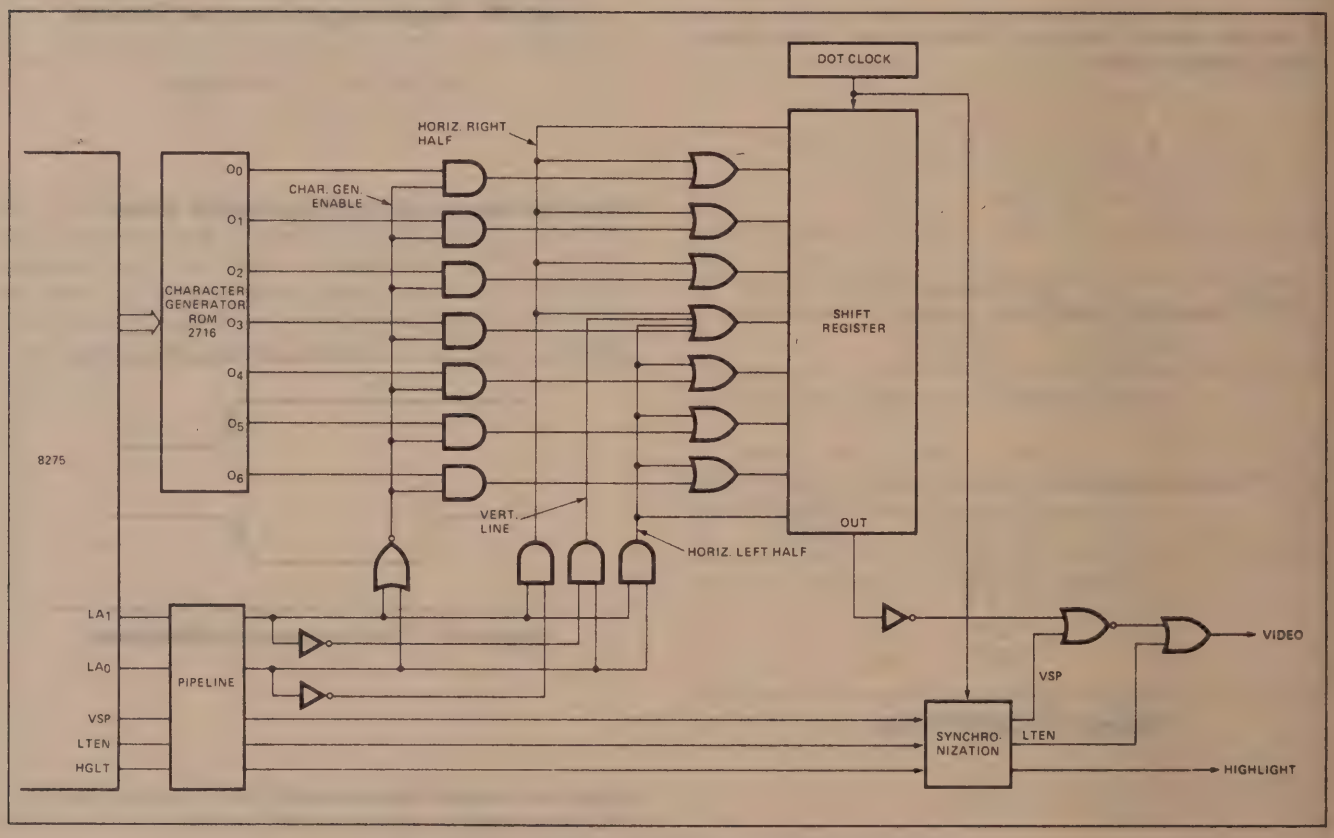


Figure 22. Typical Character Attribute Logic

Table 2. Character Attributes

Character attributes were designed to produce the following graphics:

CHARACTER ATTRIBUTE CODE "CCCC"		OUTPUTS LA_1	LA_0	VSP	LTEN	SYMBOL	DESCRIPTION
0000	Above Underline	0	0	1	0		Top Left Corner
	Underline	1	0	0	0		
	Below Underline	0	1	0	0		
0001	Above Underline	0	0	1	0		Top Right Corner
	Underline	1	1	0	0		
	Below Underline	0	1	0	0		
0010	Above Underline	0	1	0	0		Bottom Left Corner
	Underline	1	0	0	0		
	Below Underline	0	0	1	0		
0011	Above Underline	0	1	0	0		Bottom Right Corner
	Underline	1	1	0	0		
	Below Underline	0	0	1	0		
0100	Above Underline	0	0	1	0		Top Intersect
	Underline	0	0	0	1		
	Below Underline	0	1	0	0		
0101	Above Underline	0	1	0	0		Right Intersect
	Underline	1	1	0	0		
	Below Underline	0	1	0	0		
0110	Above Underline	0	1	0	0		Left Intersect
	Underline	1	0	0	0		
	Below Underline	0	1	0	0		
0111	Above Underline	0	1	0	0		Bottom Intersect
	Underline	0	0	0	1		
	Below Underline	0	0	1	0		
1000	Above Underline	0	0	1	0		Horizontal Line
	Underline	0	0	0	1		
	Below Underline	0	0	1	0		
1001	Above Underline	0	1	0	0		Vertical Line
	Underline	0	1	0	0		
	Below Underline	0	1	0	0		
1010	Above Underline	0	1	0	0		Crossed Lines
	Underline	0	0	0	1		
	Below Underline	0	1	0	0		
1011	Above Underline	0	0	0	0		Not Recommended *
	Underline	0	0	0	0		
	Below Underline	0	0	0	0		
1100	Above Underline	0	0	1	0		Special Codes
	Underline	0	0	1	0		
	Below Underline	0	0	1	0		
1101	Above Underline						Illegal
	Underline		Undefined				
	Below Underline						
1110	Above Underline						Illegal
	Underline		Undefined				
	Below Underline						
1111	Above Underline						Illegal
	Underline		Undefined				
	Below Underline						

*Character Attribute Code 1011 is not recommended for normal operation. Since none of the attribute outputs are active, the character Generator will not be disabled, and an indeterminate character will be generated.

Character Attribute Codes 1101, 1110, and 1111 are illegal.

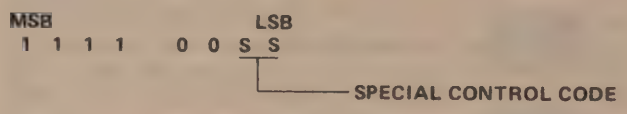
Blinking is active when B = 1.

Highlight is active when H = 1.

Special Codes

Four special codes are available to help reduce memory, software, or DMA overhead.

Special Control Character



S S	FUNCTION
0 0	End of Row
0 1	End of Row-Stop DMA
1 0	End of Screen
1 1	End of Screen-Stop DMA

The End of Row Code (00) activates VSP and holds it to the end of the line.

The End of Row-Stop DMA Code (01) causes the DMA Control Logic to stop DMA for the rest of the row when it is written into the Row Buffer. It affects the display in the same way as the End of Row Code (00).

The End of Screen Code (10) activates VSP and holds it to the end of the frame.

The End of Screen-Stop DMA Code (11) causes the DMA Control Logic to stop DMA for the rest of the frame when it is written into the Row Buffer. It affects the display in the same way as the End of Screen Code (10).

If the Stop DMA feature is not used, all characters after an End of Row character are ignored, except for the End of Screen character, which operates normally. All characters after an End of Screen character are ignored.

Note: If a Stop DMA character is not the last character in a burst or row, DMA is not stopped until after the next character is read. In this situation, a dummy character must be placed in memory after the Stop DMA character.

Field Attributes

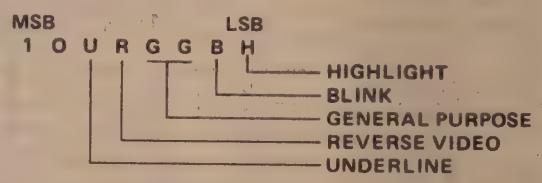
The field attributes are control codes which affect the visual characteristics for a field of characters, starting at the character following the code up to, and including, the character which precedes the *next* field attribute code, or up to the end of the frame. The field attributes are reset during the vertical retrace interval.

There are six field attributes:

1. *Blink* — Characters following the code are caused to blink by activating the Video Suppression output (VSP). The blink frequency is equal to the screen refresh frequency divided by 32.
2. *Highlight* — Characters following the code are caused to be highlighted by activating the Highlight output (HGLT).
3. *Reverse Video* — Characters following the code are caused to appear with reverse video by activating the Reverse Video output (RVV).
4. *Underline* — Characters following the code are caused to be underlined by activating the Light Enable output (LTEN).

5,6. *General Purpose* — There are two additional 8275 outputs which act as general purpose, independently programmable field attributes. GPA_{0-1} are active high outputs.

Field Attribute Code



H = 1 FOR HIGHLIGHTING
B = 1 FOR BLINKING
R = 1 FOR REVERSE VIDEO
U = 1 FOR UNDERLINE
GG = GPA_1, GPA_0

*More than one attribute can be enabled at the same time. If the blinking and reverse video attributes are enabled simultaneously, only the reversed characters will blink.

The 8275H can be programmed to provide visible or invisible field attribute characters.

If the 8275H is programmed in the visible field attribute mode, all field attributes will occupy a position on the screen. They will appear as blanks caused by activation of the Video Suppression output (VSP). The chosen visual attributes are activated after this blanked character.

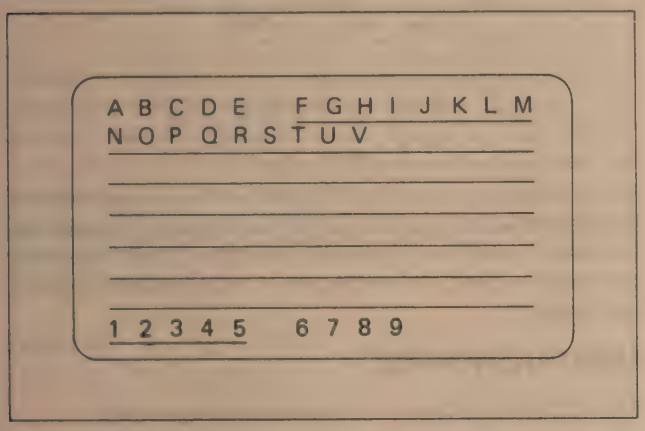


Figure 23. Example of the Visible Field Attribute Mode (Underline Attribute)

If the 8275H is programmed in the invisible field attribute mode, the 8275H FIFO is activated.

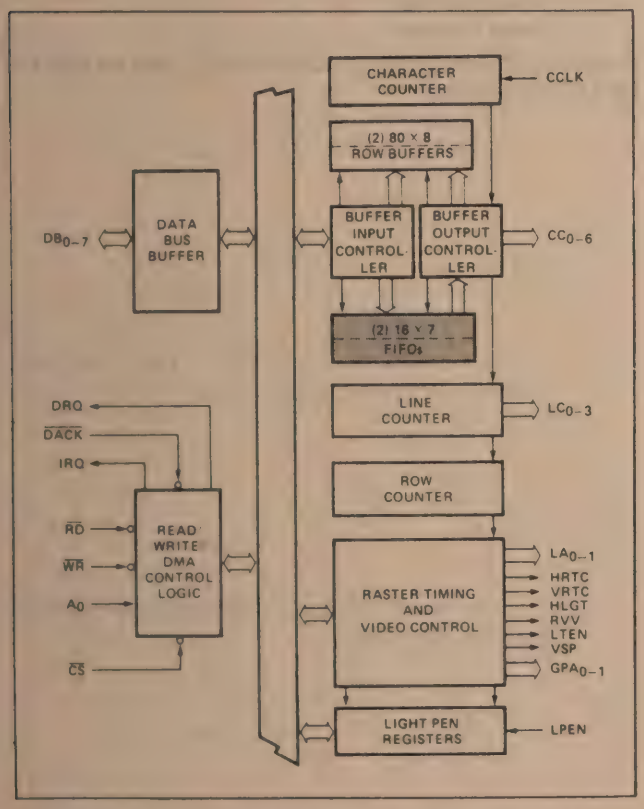


Figure 24. Block Diagram Showing FIFO Activation

Each row buffer has a corresponding FIFO. These FIFOs are 16 characters by 7 bits in size.

When a field attribute is placed in the row buffer during DMA, the buffer input controller recognizes it and places the *next* character in the proper FIFO.

When a field attribute is placed in the Buffer Output Controller during display, it causes the controller to immediately put a character from the FIFO on the Character Code outputs (CC_{0-6}). The chosen Visual Attributes are also activated.

Since the FIFO is 16 characters long, no more than 16 field attribute characters may be used per line in this mode. If more are used, a bit in the status word is set and the first characters in the FIFO are written over and lost.

Note: Since the FIFO is 7 bits wide, the MSB of any characters put in it are stripped off. Therefore, a Visual Attribute or Special Code must *not* immediately follow a field attribute code. If this situation does occur, the Visual Attribute or Special Code will be treated as a normal display character.

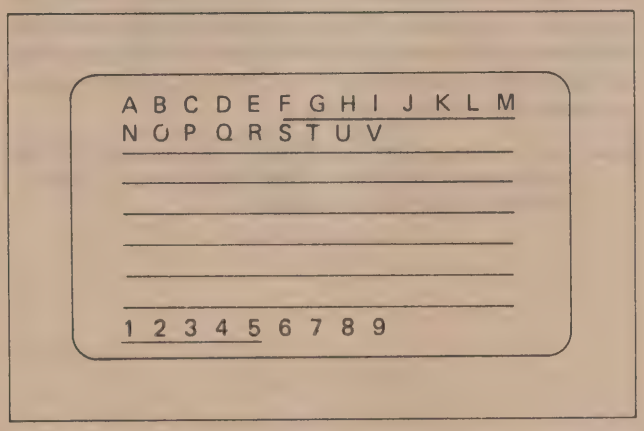


Figure 25. Example of the Invisible Field Attribute Mode (Underline Attribute)

Field and Character Attribute Interaction

Character Attribute Symbols are affected by the Reverse Video (RVV) and General Purpose (GPA_{0-1}) field attributes. They are not affected by Underline, Blink or Highlight field attributes; however, these characteristics can be programmed *individually* for Character Attribute Symbols.

Cursor Timing

The cursor location is determined by a cursor row register and a character position register which are loaded by command to the controller. The cursor can be programmed to appear on the display as:

1. a blinking underline
2. a blinking reverse video block
3. a non-blinking underline
4. a non-blinking reverse video block

The cursor blinking frequency is equal to the screen refresh frequency divided by 16.

If a non-blinking reverse video *cursor* appears in a non-blinking reverse video *field*, the cursor will appear as a normal video block.

If a non-blinking underline *cursor* appears in a non-blinking underline *field*, the cursor will not be visible.

Light Pen Detection

A light pen consists of a micro switch and a tiny light sensor. When the light pen is pressed against the CRT screen, the micro switch enables the light sensor. When the raster sweep reaches the light sensor, it triggers the light pen output.

If the output of the light pen is presented to the 8275H LPEN input, the row and character position coordinates are stored in a pair of registers. These registers can be read on command. A bit in the status word is set, indicating that the light pen signal was detected. The LPEN input must be a 0 to 1 transition for proper operation.

Note: Due to internal and external delays, the character position coordinate will be off by at least three character positions. This has to be corrected in software.

Device Programming

The 8275H has two programming registers, the Command Register (CREG) and the Parameter Register (PREG). It also has a Status Register (SREG). The Command Register can only be written into and the Status Registers can only be read from. They are addressed as follows:

A_0	OPERATION	REGISTER
0	Read	PREG
0	Write	PREG
1	Read	SREG
1	Write	CREG

The 8275H expects to receive a command and a sequence of 0 to 4 parameters, depending on the command. If the proper number of parameter bytes are not received before another command is given, a status flag is set, indicating an improper command.

INSTRUCTION SET

The 8275H instruction set consists of 8 commands.

COMMAND	NO. OF PARAMETER BYTES
Reset	4
Start Display	0
Stop Display	0
Read Light Pen	2
Load Cursor	2
Enable Interrupt	0
Disable Interrupt	0
Preset Counters	0

In addition, the status of the 8275H (SREG) can be read by the CPU at any time.

1. Reset Command:

	OPERATION	A_0	DESCRIPTION	DATA BUS MSB							LSB
Command	Write	1	Reset Command	0	0	0	0	0	0	0	0
Parameters	Write	0	Screen Comp Byte 1	S	H	H	H	H	H	H	H
	Write	0	Screen Comp Byte 2	V	V	R	R	R	R	R	R
	Write	0	Screen Comp Byte 3	U	U	U	U	L	L	L	L
	Write	0	Screen Comp Byte 4	M	F	C	C	Z	Z	Z	Z

Action — After the reset command is written, DMA requests stop, 8275 interrupts are disabled, and the VSP output is used to blank the screen. HRTC and VRTC continue to run. HRTC and VRTC timing are random on power-up.

As parameters are written, the screen composition is defined.

Parameter — S Spaced Rows

S	FUNCTIONS
0	Normal Rows
1	Spaced Rows

Parameter — HHHHHHH Horizontal Characters/Row

H H H H H H H	NO. OF CHARACTERS PER ROW
0 0 0 0 0 0 0	1
0 0 0 0 0 0 1	2
0 0 0 0 0 1 0	3
.	.
.	.
.	.
1 0 0 1 1 1 1	80
1 0 1 0 0 0 0	Undefined
.	.
.	.
.	.
1 1 1 1 1 1 1	Undefined

Parameter — VV Vertical Retrace Row Count

V V	NO. OF ROW COUNTS PER VRTC
0 0	1
0 1	2
1 0	3
1 1	4

Parameter — RRRRRR Vertical Rows/Frame

R R R R R R	NO. OF ROWS/FRAME
0 0 0 0 0 0	1
0 0 0 0 0 1	2
0 0 0 0 1 0	3
.	.
.	.
.	.
1 1 1 1 1 1	64

Parameter — UUUU Underline Placement

U U U U	LINE NUMBER OF UNDERLINE
0 0 0 0	1
0 0 0 1	2
0 0 1 0	3
.	.
.	.
.	.
1 1 1 1	16

Parameter — LLLL Number of Lines per Character Row

L L L L	NO. OF LINES/ROW
0 0 0 0	1
0 0 0 1	2
0 0 1 0	3
.	.
.	.
.	.
1 1 1 1	16

Parameter — M Line Counter Mode

M	LINE COUNTER MODE
0	Mode 0 (Non-Offset)
1	Mode 1 (Offset by 1 Count)

Parameter — F Field Attribute Mode

F	FIELD ATTRIBUTE MODE
0	Transparent
1	Non-Transparent

Parameter — CC Cursor Format

C C	CURSOR FORMAT
0 0	Blinking reverse video block
0 1	Blinking underline
1 0	Nonblinking reverse video block
1 1	Nonblinking underling

Parameter — ZZZZ Horizontal Retrace Count

Z Z Z Z	NO. OF CHARACTER COUNTS PER HRTC
0 0 0 0	2
0 0 0 1	4
0 0 1 0	6
.	.
.	.
.	.
1 1 1 1	32

Note: uuuu MSB determines blanking of top and bottom lines (1 = blanked, 0 = not blanked).

2. Start Display Command:

	OPERATION	A_0	DESCRIPTION	DATA BUS MSB							LSB
Command	Write	1	Start Display	0	0	1	S	S	S	B	B
No parameters											

S S S BURST SPACE CODE

S S S	NO. OF CHARACTER CLOCKS BETWEEN DMA REQUESTS
0 0 0	0
0 0 1	7
0 1 0	15
0 1 1	23
1 0 0	31
1 0 1	39
1 1 0	47
1 1 1	55

B B BURST COUNT CODE

B B	NO. OF DMA CYCLES PER BURST
0 0	1
0 1	2
1 0	4
1 1	8

Action — 8275 interrupts are enabled, DMA requests begin, video is enabled, Interrupt Enable and Video Enable status flags are set.

3. Stop Display Command:

	OPERATION	A_0	DESCRIPTION	DATA BUS MSB							LSB
Command	Write	1	Stop Display	0	1	0	0	0	0	0	0
No parameters											

Action — Disables video, interrupts remain enabled, HRTC and VRTC continue to run, Video Enable status flag is reset, and the "Start Display" command must be given to re-enable the display.

4. Read Light Pen Command

	OPERATION	A_0	DESCRIPTION	DATA BUS MSB							LSB
Command	Write	1	Read Light Pen	0	1	1	0	0	0	0	0
Parameters	Read	0	Char. Number	(Char. Position in Row)							
	Read	0	Row Number	(Row Number)							

Action — The 8275H is conditioned to supply the contents of the light pen position registers in the next two read cycles of the parameter register. Status flags are not affected.

Note: Software correction of light pen position is required.

5. Load Cursor Position:

	OPERATION	A_0	DESCRIPTION	DATA BUS MSB							LSB
Command	Write	1	Load Cursor	1	0	0	0	0	0	0	0
Parameters	Write	0	Char. Number	(Char. Position in Row)							
	Write	0	Row Number	(Row Number)							

Action — The 8275H is conditioned to place the next two parameter bytes into the cursor position registers. Status flags not affected.

6. Enable Interrupt Command:

	OPERATION	A_0	DESCRIPTION	DATA BUS MSB							LSB
Command	Write	1	Enable Interrupt	1	0	1	0	0	0	0	0
No parameters											

Action — The interrupt enable status flag is set and interrupts are enabled.

7. Disable Interrupt Command:

	OPERATION	A_0	DESCRIPTION	DATA BUS MSB							LSB
Command	Write	1	Disable Interrupt	1	1	0	0	0	0	0	0
No parameters											

Action — Interrupts are disabled and the interrupt enable status flag is reset.

8. Preset Counters Command:

	OPERATION	A_0	DESCRIPTION	DATA BUS MSB							LSB
Command	Write	1	Preset Counters	1	1	1	0	0	0	0	0
No parameters											

Action — The internal timing counters are preset, corresponding to a screen display position at the top left corner. Two character clocks are required for this operation. The counters will remain in this state until any other command is given.

This command is useful for system debug and synchronization of clustered CRT displays on a single CPU. After this command, two additional clock cycles are required before the first character of the first row is put out.

Status Flags

	OPERATION	A_0	DESCRIPTION	DATA BUS MSB LSB
Command	Read	1	Status Word	0 IE IR LP IC VE DU F0

IE – (Interrupt Enable) Set or reset by command. It enables vertical retrace interrupt. It is automatically set by a "Start Display" command and reset with the "Reset" command.

IR – (Interrupt Request) This flag is set at the beginning of display of the last row of the frame if the interrupt enable flag is set. It is reset after a status read operation.

LP – This flag is set when the light pen input (LPEN) is activated and the light pen registers have been loaded. This flag is automatically reset after a status read.

IC – (Improper Command) This flag is set when a command parameter string is too long or too short. The flag is automatically reset after a status read.

VE – (Video Enable) This flag indicates that video operation of the CRT is enabled. This flag is set on a "Start Display" command, and reset on a "Stop Display" or "Reset" command.

DU – (DMA Underrun) This flag is set whenever a data underrun occurs during DMA transfers. Upon detection of DU, the DMA operation is stopped and the screen is blanked until after the vertical retrace interval. This flag is reset after a status read.

FO – (FIFO Overrun) This flag is set whenever the FIFO is overrun. It is reset on a status read.

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias. . . . . 0°C to 70°C
Storage Temperature . . . . . -65°C to +150°C
Voltage On Any Pin
With Respect to Ground . . . . . -0.5V to +7V
Power Dissipation . . . . . 1 Watt

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied.*

D.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = 5V ±5%)

Symbol	Parameter	Min.	Max.	Units	Test Conditions
V_{IL}	Input Low Voltage	-0.5	0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC}+0.5V	V	
V_{OL}	Output Low Voltage		0.45	V	I_{OL} = 2.2 mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = -400 μA
I_{IL}	Input Load Current		±10	μA	V_{IN} = V_{CC} to 0V
I_{OFL}	Output Float Leakage		±10	μA	V_{OUT} = V_{CC} to 0.45V
I_{CC}	V_{CC} Supply Current		160	mA	

CAPACITANCE (T_A = 25°C, V_{CC} = GND = 0V)

Symbol	Parameter	Min.	Max.	Units	Test Conditions
C_{IN}	Input Capacitance		10	pF	f_c = 1 MHz
$C_{I/O}$	I/O Capacitance		20	pF	Unmeasured pins returned to V_{SS}.

A.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = 5.0V ±5%, GND = 0V)

Bus Parameters

READ CYCLE

Symbol	Parameter	Min.	Max.	Units	Test Conditions
t_{AR}	Address Stable Before READ	0		ns	
t_{RA}	Address Hold Time for READ	0		ns	
t_{RR}	READ Pulse Width	250		ns	
t_{RD}	Data Delay from READ		200	ns	C_L = 150 pF
t_{DF}	READ to Data Floating		100	ns	C_L = 150 pF

WRITE CYCLE

Symbol	Parameter	Min.	Max.	Units	Test Conditions
t_{AW}	Address Stable Before WRITE	0		ns	
t_{WA}	Address Hold Time for WRITE	0		ns	
t_{WW}	WRITE Pulse Width	250		ns	
t_{DW}	Data Setup Time for WRITE	150		ns	
t_{WD}	Data Hold Time for WRITE	0		ns	

A.C. CHARACTERISTICS (Continued)

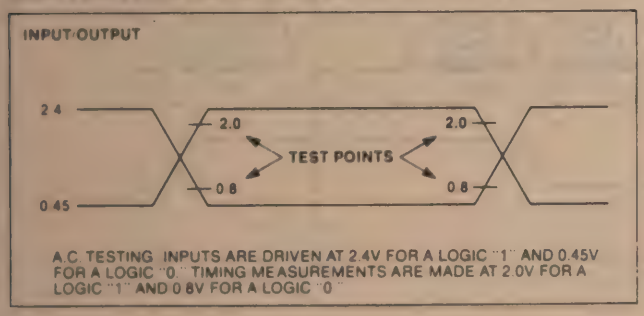
CLOCK TIMING

Symbol	Parameter	8275 Min.	8275 Max.	8275-2 Min.	8275-2 Max.	Units	Test Conditions
t_{CLK}	Clock Period	480		320		ns	
t_{KH}	Clock High	240		120		ns	
t_{KL}	Clock Low	160		120		ns	
t_{KR}	Clock Rise	5	30	5	30	ns	
t_{KF}	Clock Fall	5	30	5	30	ns	

OTHER TIMING

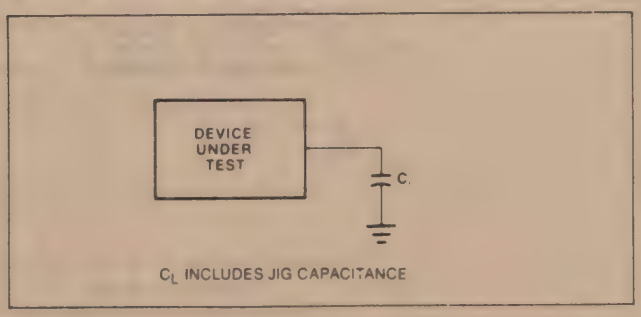
Symbol	Parameter	8275 Min.	8275 Max.	8275-2 Min.	8275-2 Max.	Units	Test Conditions
t_{CC}	Character Code Output Delay		150		150	ns	C_L = 50 pF
t_{HR}	Horizontal Retrace Output Delay		200		150	ns	C_L = 50 pF
t_{LC}	Line Count Output Delay		400		250	ns	C_L = 50 pF
t_{AT}	Control/Attribute Output Delay		275		250	ns	C_L = 50 pF
t_{VR}	Vertical Retrace Output Delay		275		250	ns	C_L = 50 pF
t_{RI}	IRQ↓ from RD↑		250		250	ns	C_L = 50 pF
t_{WQ}	DRQ↑ from WR↑		250		250	ns	C_L = 50 pF
t_{RQ}	DRQ↓ from WR↓		200		200	ns	C_L = 50 pF
t_{LR}	DACK↓ to WR↓	0		0		ns	
t_{RL}	WR↑ to DACK↑	0		0		ns	
t_{PR}	LPEN Rise		50		50	ns	
t_{PH}	LPEN Hold	100		100		ns	
t_{DI}	DACK Inactive Period	120				ns	

A.C. TESTING INPUT, OUTPUT WAVEFORM



A.C. TESTING INPUTS ARE DRIVEN AT 2.4V FOR A LOGIC "1" AND 0.45V FOR A LOGIC "0." TIMING MEASUREMENTS ARE MADE AT 2.0V FOR A LOGIC "1" AND 0.8V FOR A LOGIC "0"

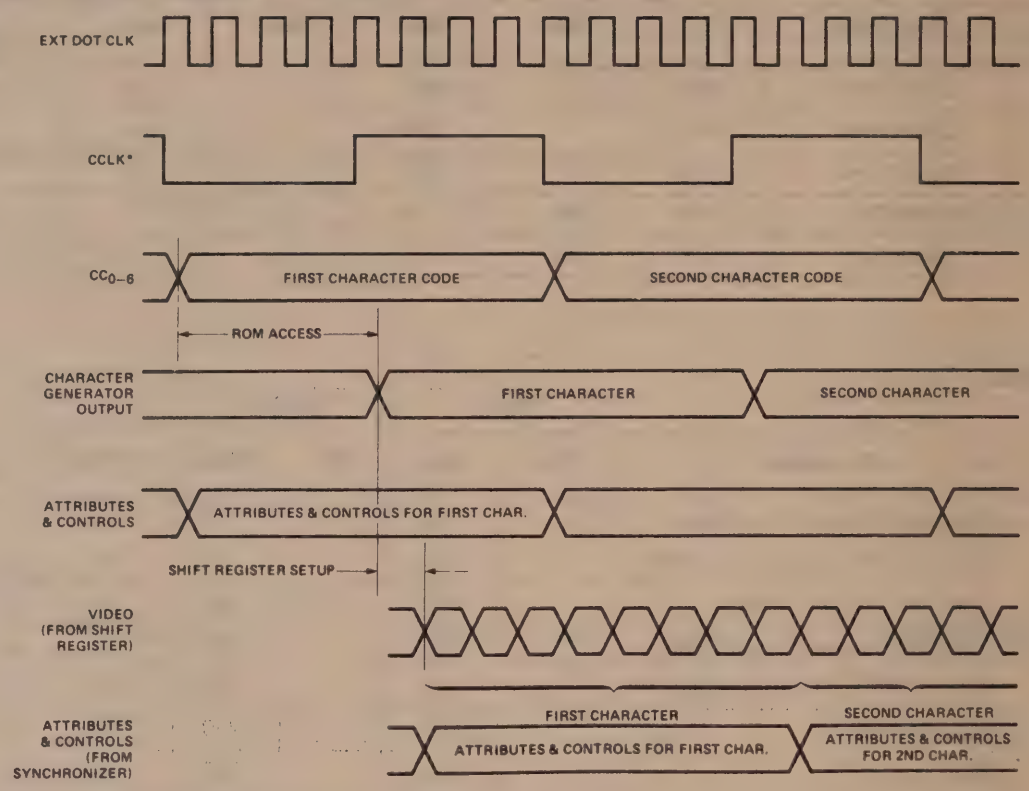
A.C. TESTING LOAD CIRCUIT



C_L INCLUDES JIG CAPACITANCE

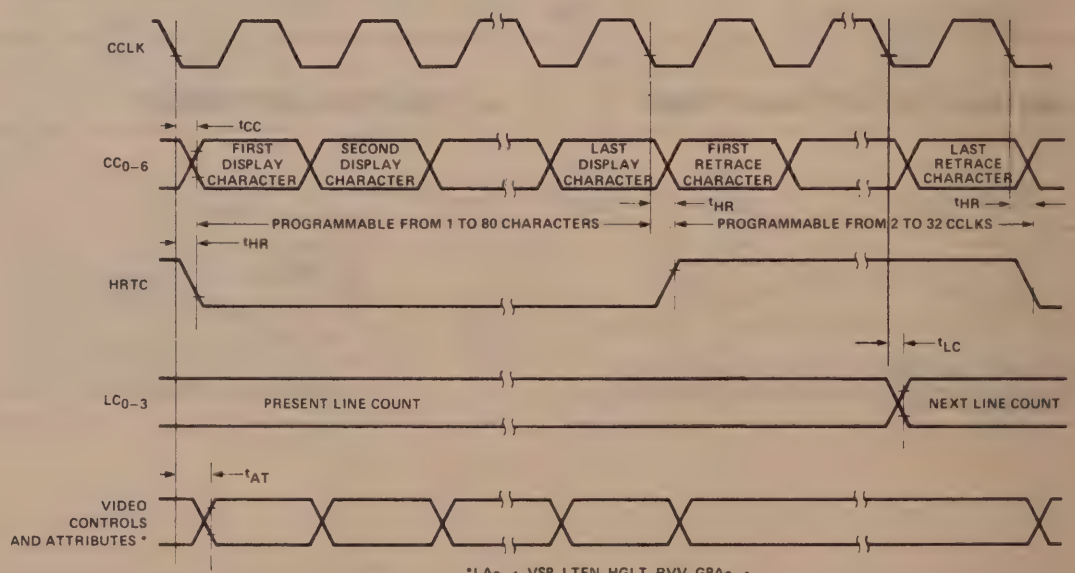
WAVEFORMS

TYPICAL DOT LEVEL TIMING



*CCLK IS A MULTIPLE OF THE DOT CLOCK AND AN INPUT TO THE 8275.

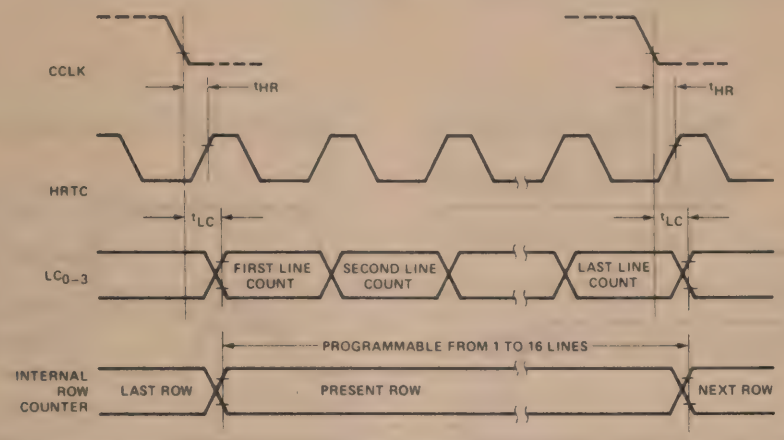
LINE TIMING



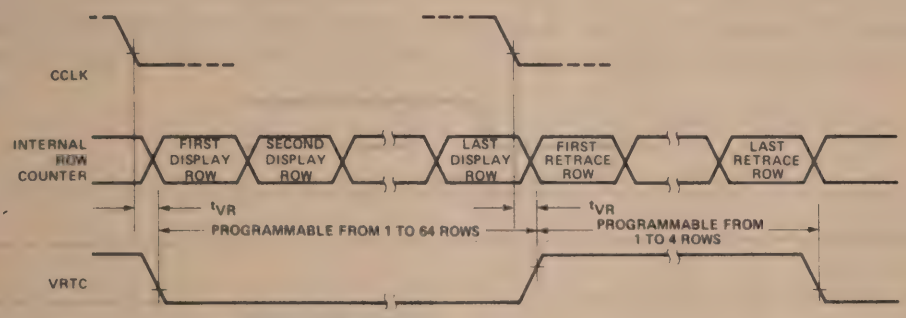
*LA0–1, VSP, LTEN, HGLT, RVV, GPA0–1

WAVEFORMS (Continued)

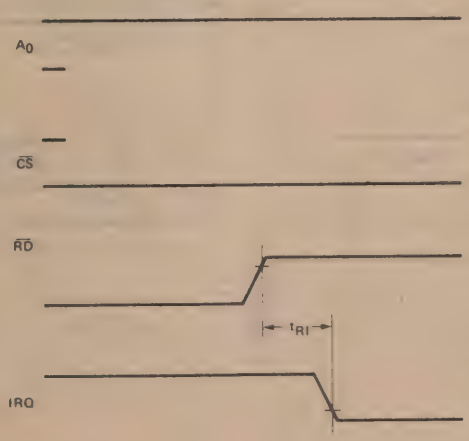
ROW TIMING



FRAME TIMING

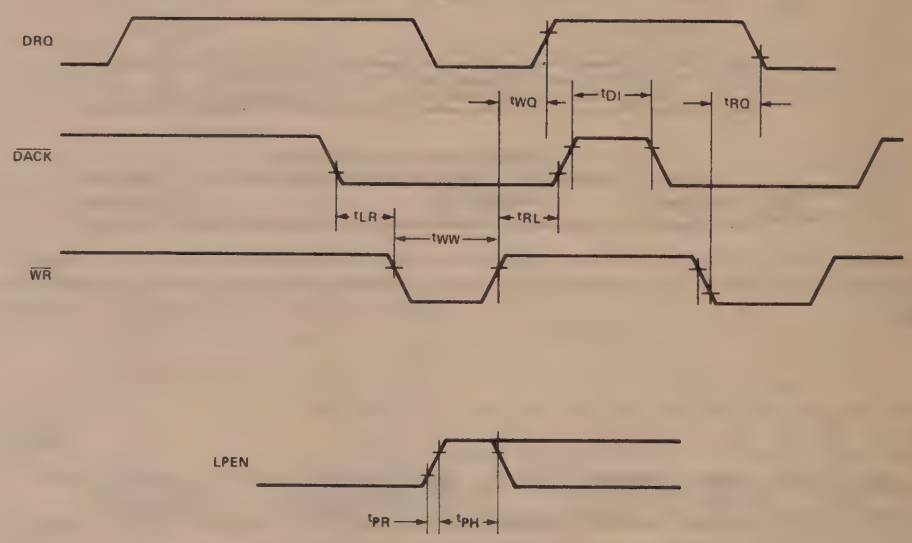


INTERRUPT TIMING

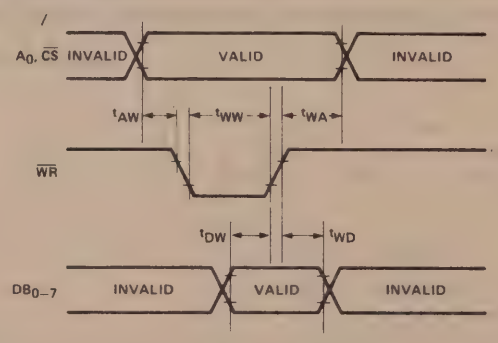


WAVEFORMS (Continued)

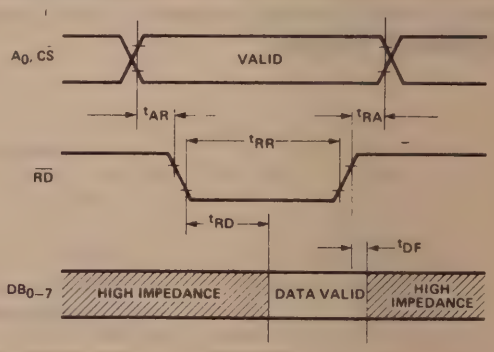
DMA TIMING



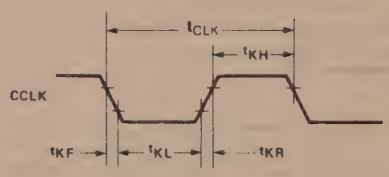
WRITE TIMING



READ TIMING



CLOCK TIMING



intel

8276H
SMALL SYSTEM CRT CONTROLLER

- **Programmable Screen and Character Format**
- **6 Independent Visual Field Attributes**
- **Cursor Control (4 Types)**
- **MCS-51®, MCS-85®, iAPX 86, and iAPX 88 Compatible**
- **Dual Row Buffers**
- **Single +5V Supply**
- **40-Pin Package**
- **3 MHz Clock with 8276-2**
- **High Performance HMOS-II**

The Intel 8276H Small System CRT Controller is a single chip device intended to interface CRT raster scan displays with Intel microcomputers in minimum device-count systems. Its primary function is to refresh the display by buffering character information from main memory and keeping track of the display position of the screen. The flexibility designed into the 8276H will allow simple interface to almost any raster scan CRT display. It can be used with the 8051 Single Chip Microcomputer for a minimum IC count design. It is manufactured on Intel's advanced HMOS-II process.

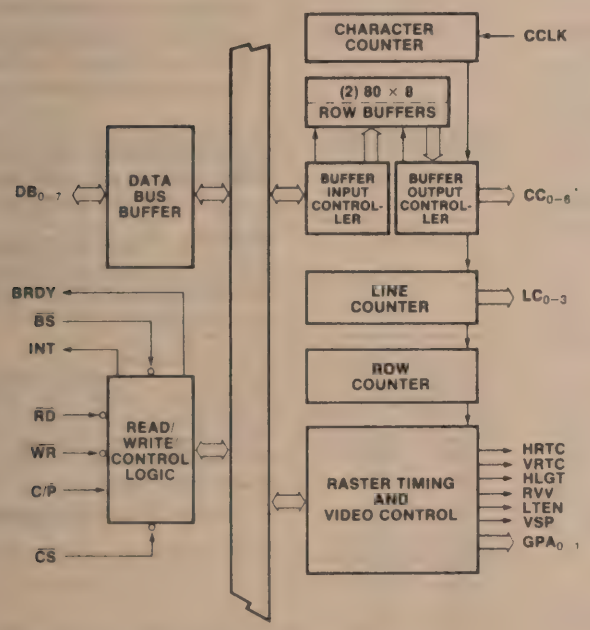


Figure 1. Block Diagram

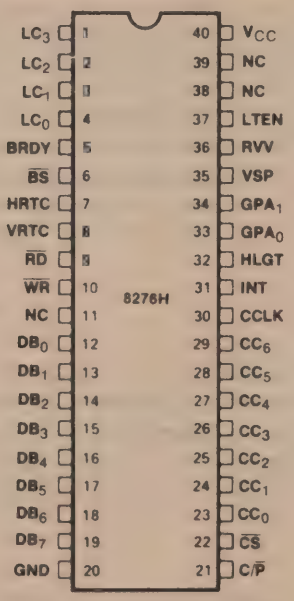


Figure 2. Pin Configuration

Intel Corporation Assumes No Responsibility for the Use of Any Circuitry Other Than Circuitry Embodied in an Intel Product. No Other Circuit Patent Licenses are Implied.
©INTEL CORPORATION, 1984
ORDER NUMBER: 210668-002

Table 1. Pin Descriptions

Symbol	Pin No.	Type	Name and Function
LC_3 LC_2 LC_1 LC_0	1 2 3 4	O	**Line count.** Output from the line counter which is used to address the character generator for the line positions on the screen.
BRDY	5	O	**Buffer ready.** Output signal indicating that a Row Buffer is ready for loading of character data.
$\overline{BS}$	6	I	**Buffer select.** Input signal enabling $\overline{WR}$ for character data into the Row Buffers.
HRTC	7	O	**Horizontal retrace.** Output signal which is active during the programmed horizontal retrace interval. During this period the VSP output is high and the LTEN output is low.
VRTC	8	O	**Vertical retrace.** Output signal which is active during the programmed vertical retrace interval. During this period the VSP output is high and the LTEN output is low.
$\overline{RD}$	9	I	**Read input.** A control signal to read registers.
$\overline{WR}$	10	I	**Write input.** A control signal to write commands into the control registers or write data into the row buffers.
NC	11		**No connection.**
DB_0 DB_1 DB_2 DB_3 DB_4 DB_5 DB_6 DB_7	12 13 14 15 16 17 18 19	I/O	**Bidirectional data bus.** Three-state lines. The outputs are enabled during a read of the C or P ports.
Ground	20		**Ground.**
V_{CC}	40		**+5V power supply.**
NC	39		**No connection.**
NC	38		**No connection.**
LTEN	37	O	**Light enable.** Output signal used to enable the video signal to the CRT. This output is active at the programmed underline cursor position, and at positions specified by attribute codes.
RVV	36	O	**Reverse video.** Output signal used to activate the CRT circuitry to reverse the video signal. This output is active at the cursor position if a reverse video block cursor is programmed or at the positions specified by the field attribute codes.
VSP	35	O	**Video suppression.** Output signal used to blank the video signal to the CRT. This output is active: — during the horizontal and vertical retrace intervals. — at the top and bottom lines of rows if underline is programmed to be number 8 or greater. — when an end of row or end of screen code is detected. — when a Row Buffer underrun occurs. — at regular intervals (1/16 frame frequency for cursor, 1/32 frame frequency for attributes)—to create blinking displays as specified by cursor or field attribute programming.
GPA_1 GPA_0	34 33	O	**General purpose attribute codes.** Outputs which are enabled by the general purpose field attribute codes.
HLGT	32	O	**Highlight.** Output signal used to intensify the display at particular positions on the screen as specified by the field attribute codes.
INT	31	O	**Interrupt output.**
CCLK	30	I	**Character clock** (from dot/timing logic).
CC_6 CC_5 CC_4 CC_3 CC_2 CC_1 CC_0	29 28 27 26 25 24 23	O	**Character codes.** Output from the row buffers used for character selection in the character generator.
$\overline{CS}$	22	I	**Chip select.** Enables $\overline{RD}$ of status or $\overline{WR}$ of command or parameters.
C/ $\overline{P}$	21	I	**Port address.** A high input on this pin selects the "C" port or command registers and a low input selects the "P" port or parameter registers.

FUNCTIONAL DESCRIPTION

Data Bus Buffer

This 3-state, bidirectional, 8-bit buffer is used to interface the 8276H to the system Data Bus.

This functional block accepts inputs from the System Control Bus and generates control signals for overall device operation. It contains the Command, Parameter, and Status Registers that store the various control formats for the device functional definition.

C/$\overline{P}$	OPERATION	REGISTER
0	Read	RESERVED
0	Write	PARAMETER
1	Read	STATUS
1	Write	COMMAND

$\overline{RD}$ (READ)
A "low" on this input informs the 8276H that the CPU is reading status information from the 8276H.

$\overline{WR}$ (WRITE)
A "low" on this input informs the 8276H that the CPU is writing data or control words to the 8276H.

$\overline{CS}$ (CHIP SELECT)
A "low" on this input selects the 8276H for $\overline{RD}$ or $\overline{WR}$ of Commands, Status, and Parameters.

BRDY (BUFFER READY)
A "high" on this output indicates that the 8276H is ready to receive character data.

$\overline{BS}$ (BUFFER SELECT)
A "low" on this input enables $\overline{WR}$ of character data to the 8276H row buffers.

INT (INTERRUPT)
A "high" on this output informs the CPU that the 8276H needs interrupt service.

C/$\overline{P}$	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	$\overline{BS}$	
0	0	1	0	1	Reserved
0	1	0	0	1	Write 8276H Parameter
1	0	1	0	1	Read 8276H Status
1	1	0	0	1	Write 8276H Command
X	1	0	1	0	Write 8276H Row Buffer
X	1	1	X	X	High Impedance
X	X	X	1	1	High Impedance

Character Counter

The Character Counter is a programmable counter that is used to determine the number of characters to be displayed per row and the length of the horizontal retrace interval. It is driven by the CCLK (Character Clock) input, which should be derived from the external dot clock.

Line Counter

The Line Counter is a programmable counter that is used to determine the number of horizontal lines (Raster Scans) per character row. Its outputs are used to address the external character generator.

Row Counter

The Row Counter is a programmable counter that is used to determine the number of character rows to be displayed per frame and length of the vertical retrace interval.

Raster Timing and Video Controls

The Raster Timing circuitry controls the timing of the HRTC (Horizontal Retrace) and VRTC (Vertical Retrace) outputs. The Video Control circuitry controls the generation of HGLT (Highlight), RVV (Reverse Video), LTEN (Light Enable), VSP (Video Suppress), and GPA_{0-1} (General Purpose Attribute) outputs.

Row Buffers

The Row Buffers are two 80-character buffers. They are filled from the microcomputer system memory with the character codes to be displayed. While one row buffer is displaying a row of characters, the other is being filled with the next row of characters.

Buffer Input/Output Controllers

The Buffer Input/Output Controllers decode the characters being placed in the row buffers. If the character is a field attribute or special code, they control the appropriate action. (Example: A "Highlight" field attribute will cause the Buffer Output Controller to activate the HGLT output.)

ORDER NUMBER: 210668-002

SYSTEM OPERATION

The 8276H is programmable to a large number of different display formats. It provides raster timing, display row buffering, visual attribute decoding and cursor timing.

It is designed to interface with standard character generators for dot matrix decoding. Dot level timing must be provided by external circuitry.

General Systems Operational Description

Display characters are retrieved from memory and displayed on a row-by-row basis. The 8276H has two row buffers. While one row buffer is being used for display, the other is being filled with the next row of characters to be displayed. The number of display characters per row and the number of character rows per frame are software programmable, providing easy interface to most CRT displays. (See Programming Section.)

The 8276H uses BRDY to request character data to fill the row buffer that is not being used for display.

The 8276H displays character rows one scan line at a time. The number of scan lines per character row, the underline position, and blanking of top and bottom lines are programmable. (See Programming Section.)

The 8276H provides special Control Codes which can be used to minimize overhead. It also provides Visual Attribute Codes to cause special action on the screen without the use of the character generator. (See Visual Attributes Section.)

The 8276H also controls raster timing. This is done by generating Horizontal Retrace (HRTC) and Vertical Retrace (VRTC) signals. The timing of these signals is also programmable.

The 8276H can generate a cursor. Cursor location and format are programmable. (See Programming Section.)

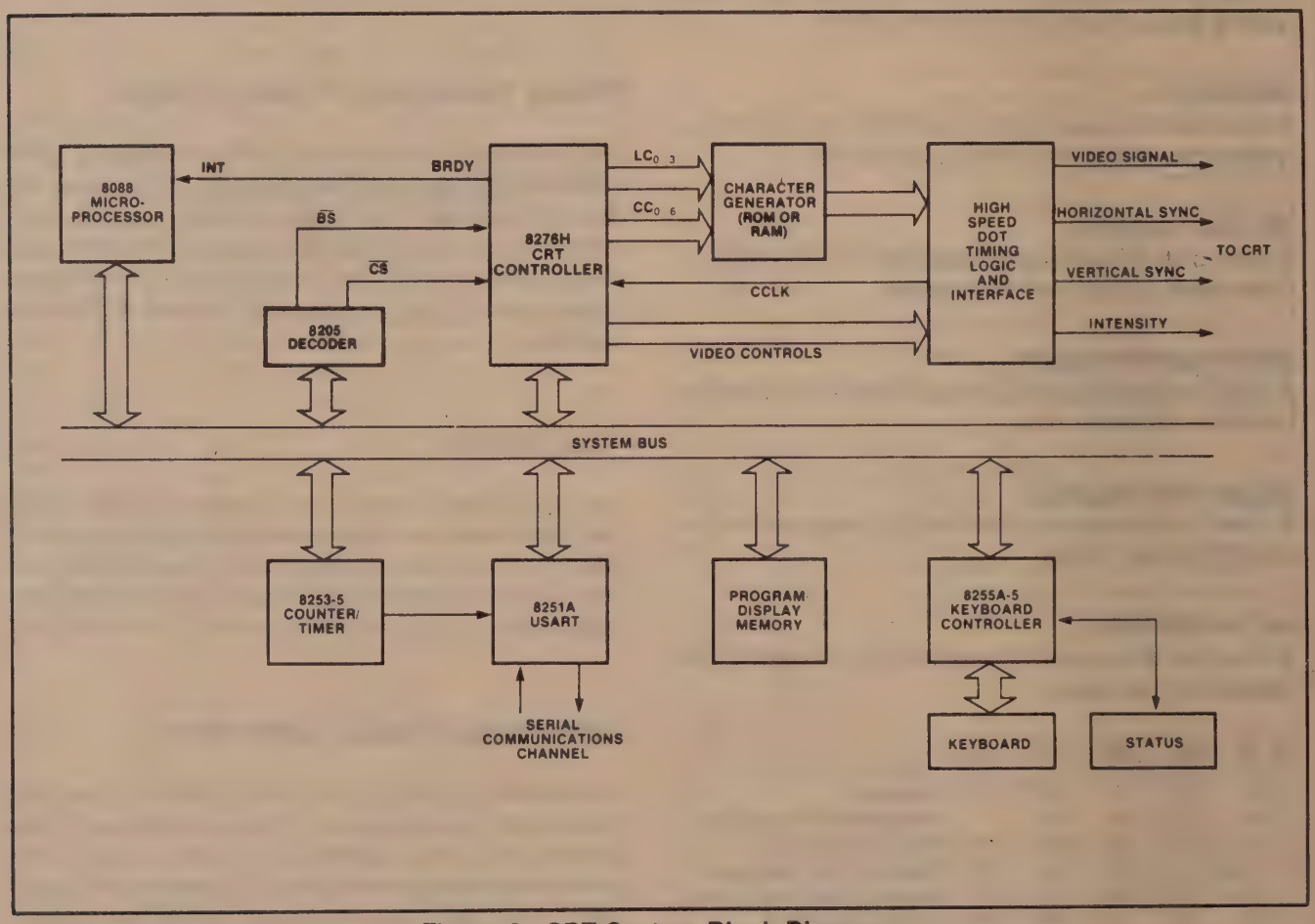


Figure 3. CRT System Block Diagram

AFN-00224B

Figure 4. Display Of A Character Row

Display Row Buffering

Before the start of a frame, the 8276H uses BRDY and $\overline{BS}$ to fill one row buffer with characters.

When the first horizontal sweep is started, character codes are output to the character generator from the row buffer just filled. Simultaneously, the other row buffer is filled with the next row of characters.

After all the lines of the character row are scanned, the buffers are swapped and the same procedure is followed for the next row.

This process is repeated until all of the character rows are displayed.

Row Buffering allows the CPU access to the display memory at all times except during Buffer Loading (about 25%). This compares favorably to alternative approaches which restrict CPU access to the display memory to occur only during horizontal and vertical retrace intervals (80% of the bus time is used to refresh the display.)

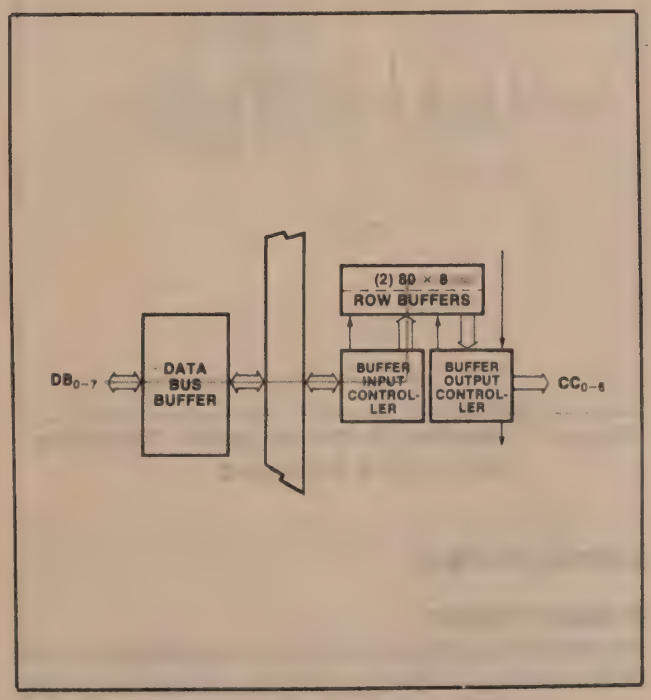


Figure 5. First Row Buffer Filled

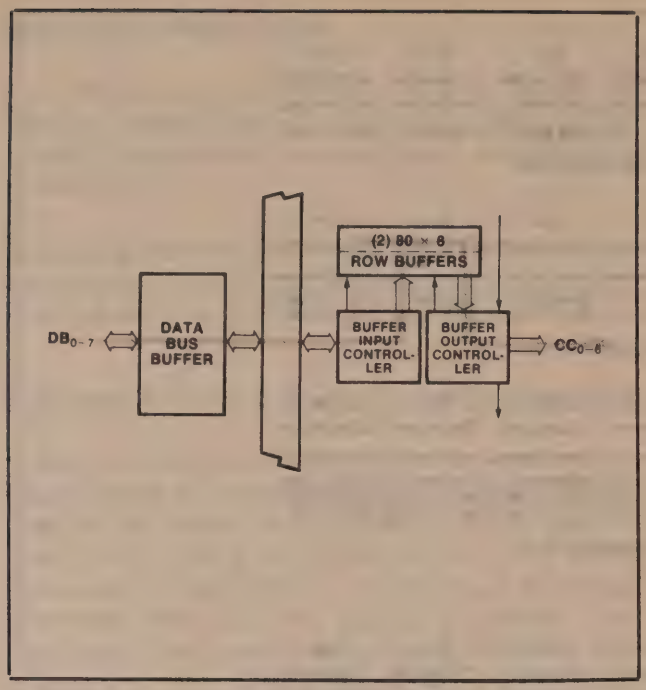


Figure 6. Second Row Buffer Filled, First Row Displayed

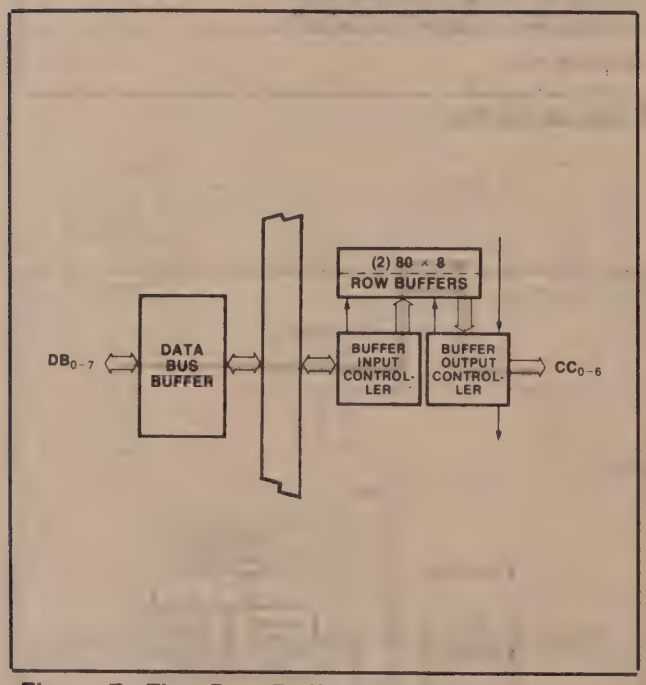


Figure 7. First Row Buffer Filled With Third Row, Second Row Displayed

Display Format

SCREEN FORMAT

The 8276H can be programmed to generate from 1 to 80 characters per row, and from 1 to 64 rows per frame.

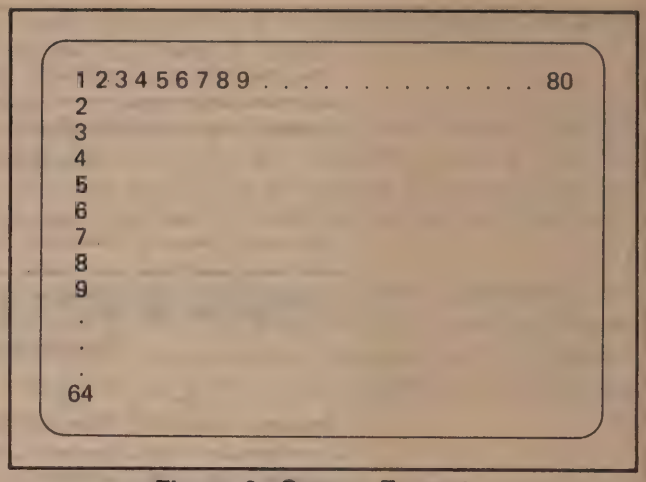


Figure 8. Screen Format

The 8276H can also be programmed to blank alternate rows. In this mode, the first row is displayed, the second blanked, the third displayed, etc. Display data is not requested for the blanked rows.

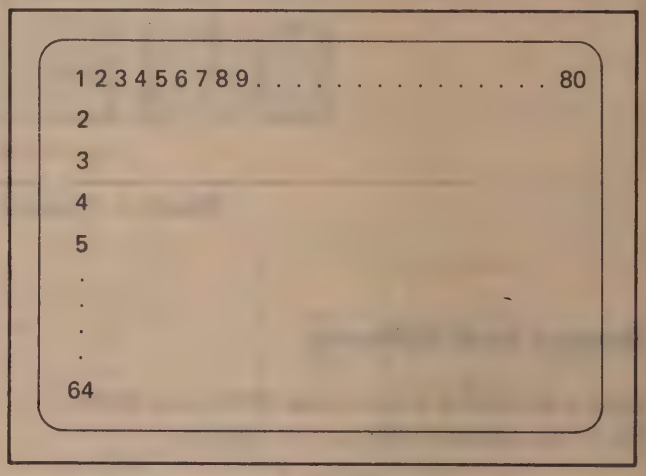


Figure 9. Blank Alternate Rows Mode

ROW FORMAT

The 8276H is designed to hold the line count stable while outputting the appropriate character codes during each horizontal sweep. The line count is incremented during horizontal retrace and the whole row of character codes are output again during the next sweep. This is continued until the entire character row is displayed.

The number of lines (horizontal sweeps) per character row is programmable from 1 to 16.

The output of the line counter can be programmed to be in one of two modes.

In mode 0, the output of the line *counter* is the same as the line *number*.

In mode 1, the line *counter* is offset by one from the line *number*.

Note: In mode 1, while the *first* line (line number 0) is being displayed, the *last* count is output by the line counter (see examples).

Line Number	Line Counter Mode 0	Line Counter Mode 1
0	0000	1111
1	0001	0000
2	0010	0001
3	0011	0010
4	0100	0011
5	0101	0100
6	0110	0101
7	0111	0110
8	1000	0111
9	1001	1000
10	1010	1001
11	1011	1010
12	1100	1011
13	1101	1100
14	1110	1101
15	1111	1110

Figure 10. Example of a 16-Line Format

Line Number	Line Counter Mode 0	Line Counter Mode 1
0	0000	1001
1	0001	0000
2	0010	0001
3	0011	0010
4	0100	0011
5	0101	0100
6	0110	0101
7	0111	0110
8	1000	0111
9	1001	1000

Figure 11. Example of a 10-Line Format

Mode 0 is useful for character generators that leave address zero blank and start at address 1. Mode 1 is useful for character generators which start at address zero.

Underline placement is also programmable (from line *number* 0 to 15). This is independent of the line *counter* mode.

If the line *number* of the underline is greater than 7 (line *number* MSB = 1), then the top and bottom lines will be blanked.

Line Number	Line Counter Mode 0	Line Counter Mode 1
0	0000	1011
1	0001	0000
2	0010	0001
3	0011	0010
4	0100	0011
5	0101	0100
6	0110	0101
7	0111	0110
8	1000	0111
9	1001	1000
10	1010	1001
11	1011	1010

Top and Bottom Lines are Blanked

Figure 12. Underline in Line Number 10

If the line *number* of the underline is less than or equal to 7 (line *number* MSB = 0), then the top and bottom lines will *not* be blanked.

Line Number	Line Counter Mode 0	Line Counter Mode 1
0	0000	0111
1	0001	0000
2	0010	0001
3	0011	0010
4	0100	0011
5	0101	0100
6	0110	0101
7	0111	0110

Top and Bottom Lines are not Blanked

Figure 13. Underline in Line Number 7

If the line *number* of the underline is greater than the maximum number of lines, the underline will not appear.

Blanking is accomplished by the VSP (Video Suppression) signal. Underline is accomplished by the LTEN (Light Enable) signal.

DOT FORMAT

Dot width and character width are dependent upon the external timing and control circuitry.

Dot level timing circuitry should be designed to accept the parallel output of the character generator and shift it out serially at the rate required by the CRT display.

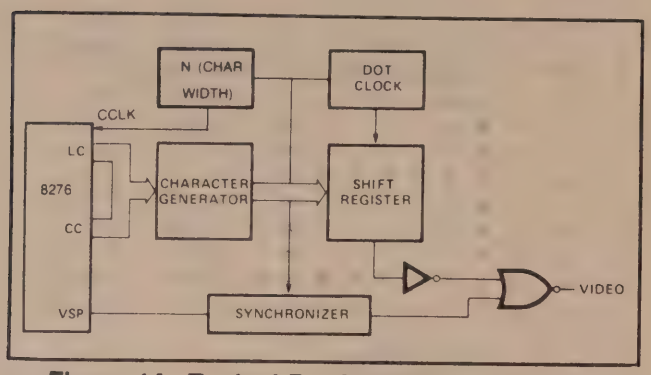


Figure 14. Typical Dot Level Block Diagram

Dot width is a function of dot clock frequency.

Character width is a function of the character generator width.

Horizontal character spacing is a function of the shift register length.

Note: Video control and timing signals must be synchronized with the video signal due to the character generator access delay.

Raster Timing

The character counter is driven by the character clock input (CCLK). It counts out the characters being displayed (programmable from 1 to 80). It then causes the line counter to increment, and it starts counting out the horizontal retrace interval (programmable from 2 to 32). This process is constantly repeated.

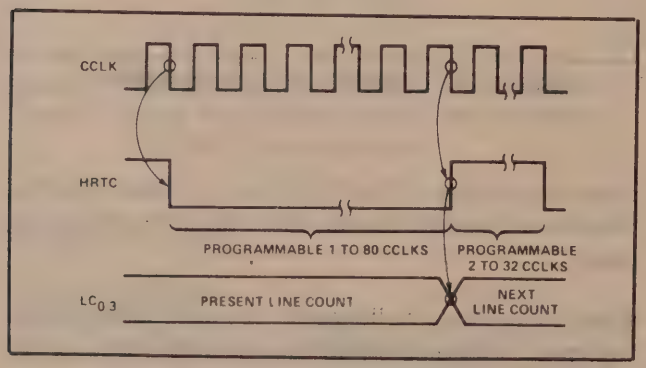


Figure 15. Line Timing

The line counter is driven by the character counter. It is used to generate the line address outputs (LC_{0-3}) for the character generator. After it counts all of the lines in a character row (programmable from 1 to 16), it increments the row counter, and starts over again. (See Character Format Section for detailed description of Line Counter functions.)

The row counter is an internal counter driven by the line counter. It controls the functions of the row buffers and counts the number of character rows displayed.

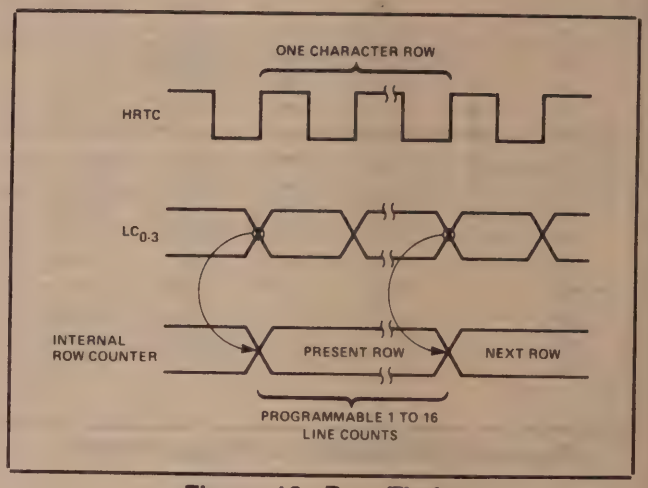


Figure 16. Row Timing

After the row counter counts all of the rows in a frame (programmable from 1 to 64), it starts counting out the vertical retrace interval (programmable from 1 to 4).

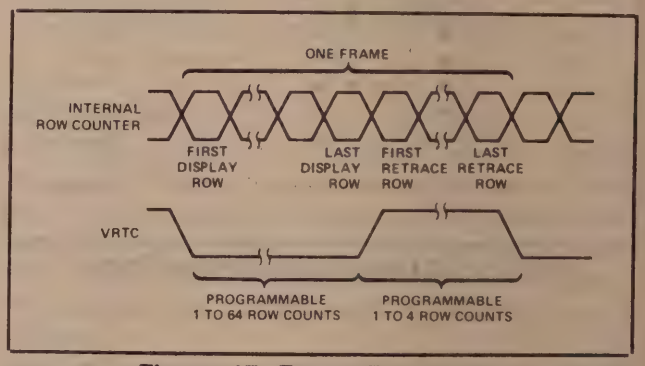


Figure 17. Frame Timing

The Video Suppression Output (VSP) is active during horizontal and vertical retrace intervals.

Dot level timing circuitry must synchronize these outputs with the video signal to the CRT Display.

Interrupt Timing

The 8276H can be programmed to generate an interrupt request at the end of each frame. If the 8276H interrupt enable flag is set, an interrupt request will occur at the *beginning* of the *last display row.*

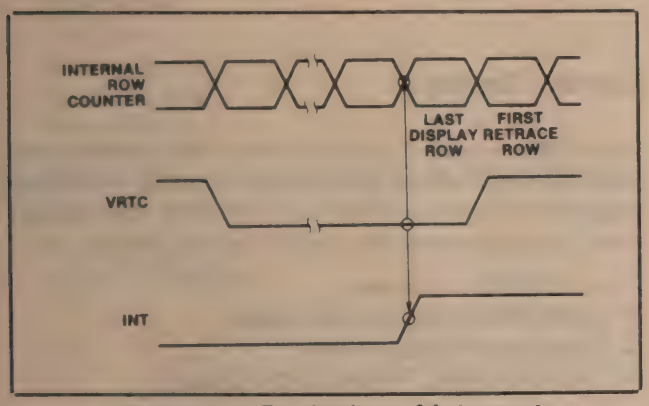


Figure 18. Beginning of Interrupt

INT will go inactive after the status register is read.

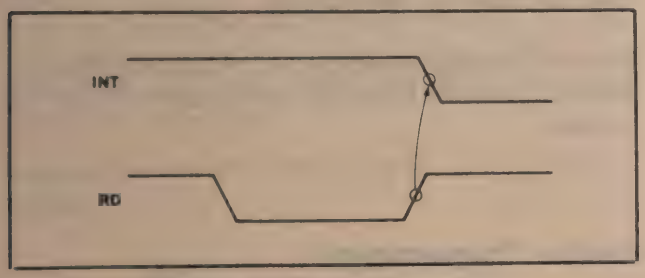


Figure 19. End of Interrupt

A reset command will also cause INT to go inactive, but this is not recommended during normal service.

Note: Upon power-up, the 8276H Interrupt Enable Flag may be set. As a result, the user's cold start routine should write a reset command to the 8276H before system interrupts are enabled.

VISUAL ATTRIBUTES AND SPECIAL CODES

The characters processed by the 8276H are 8-bit quantities. The character code outputs provide the character generator with 7 bits of address. The Most Significant Bit is the extra bit and it is used to determine if it is a normal display character (MSB = 0), or if it is a Field Attribute or Special Code (MSB = 1).

Special Codes

Four special codes are available to help reduce bus usage.

SPECIAL CONTROL CHARACTER

MSB LSB

1 1 1 1 0 0 S S — SPECIAL CONTROL CODE

S	S	FUNCTION
0	0	End of Row
0	1	End of Row-Stop Buffer Loading
1	0	End of Screen
1	1	End of Screen-Stop Buffer Loading

The End of Row Code (00) activates VSP and holds it to the end of the line.

The End of Row-Stop Buffer Loading (BRDY) Code (01) causes the Buffer Loading Control Logic to stop buffer loading for the rest of the row upon being written into the Row Buffer. It affects the display in the same way as the End of Row Code (00).

The End of Screen Code (10) activates VSP and holds it to the end of the frame.

The End of Screen-Stop Buffer Loading (BRDY) Code (11) causes the Row Buffer Control Logic to stop buffer loading for the rest of the frame upon being written. It affects the display in the same way as the End of Screen Code (10).

If the Stop Buffer Loading feature is not used, all characters after an End of Row character are ignored, except for the End of Screen character, which operates normally. All characters after an End of Screen character are ignored.

Note: If a Stop Buffer Loading is not the last character in a row, Buffer Loading is not stopped until after the next character is read. In this situation, a dummy character must be placed in memory after the Stop Buffer Loading character.

Field Attributes

The field attributes are control codes which affect the visual characteristics for a field of characters, starting at the character following the code up to, and including, the character which precedes the *next* field attribute code, or up to the end of the frame. The field attributes are reset during the vertical retrace interval.

ORDER NUMBER: 210688-002

The 8276H can be programmed to provide visible field attribute characters; all field attribute codes will occupy a position on the screen. These codes will appear as blanks caused by activation of the Video Suppression output (VSP). The chosen visual attributes are activated after this blanked character.

There are six field attributes:

1. *Blink*—Characters following the code are caused to blink by activating the Video Suppression output (VSP). The blink frequency is equal to the screen refresh frequency divided by 32.
2. *Highlight*—Characters following the code are caused to be highlighted by activating the Highlight output (HGLT).
3. *Reverse Video*—Characters following the code are caused to appear with reverse video by activating the Reverse Video output (RVV).
4. *Underline*—Characters following the code are caused to be underlined by activating the Light Enable output (LTEN).

5,6. *General Purpose*—There are two additional 8276H outputs which act as general purpose, independently programmable field attributes. GPA_{0-1} are active high outputs.

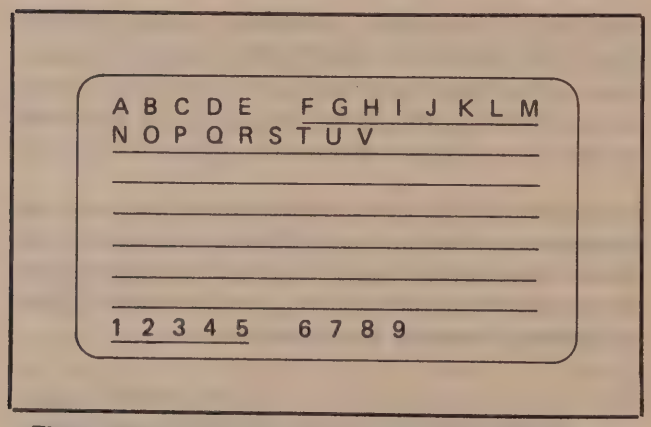
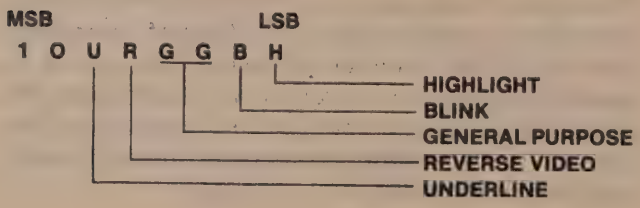


Figure 20. Example of a Visible Field Attribute (Underline Attribute)

FIELD ATTRIBUTE CODE



H = 1 FOR HIGHLIGHTING
B = 1 FOR BLINKING
R = 1 FOR REVERSE VIDEO
U = 1 FOR UNDERLINE
GG = GPA_1, GPA_0

Note: More than one attribute can be enabled at the same time. If the blinking and reverse video attributes are enabled simultaneously, only the reversed characters will blink.

Cursor Timing

The cursor location is determined by a cursor row register and a character position register which are loaded by command to the controller. The cursor can be programmed to appear on the display as:

1. a blinking underline
2. a blinking reverse video block
3. a non-blinking underline
4. a non-blinking reverse video block

The cursor blinking frequency is equal to the screen refresh frequency divided by 16.

If a non-blinking reverse video *cursor* appears in a non-blinking reverse video *field*, the cursor will appear as a normal video block.

If a non-blinking underline *cursor* appears in a non-blinking underline *field*, the cursor will not be visible.

Device Programming

The 8276H has two programming registers, the Command Register and the Parameter Register. It also has a Status Register. The Command Register can only be written into and the Status Register can only be read from. They are addressed as follows:

C/$\overline{P}$	OPERATION	REGISTER
0	Read	Reserved
0	Write	Parameter
1	Read	Status
1	Write	Command

The 8276H expects to receive a command and a sequence of 0 to 4 parameters, depending on the command. If the proper number of parameter bytes are not received before another command is given, a status flag is set, indicating an improper command.

Instruction Set

The 8276H instruction set consists of 7 commands.

COMMAND	NO. OF PARAMETER BYTES
Reset	4
Start Display	0
Stop Display	0
Load Cursor	2
Enable Interrupt	0
Disable Interrupt	0
Preset Counters	0

In addition, the status of the 8276H can be read by the CPU at any time.

1. RESET COMMAND

	OPERATION	C/P	DESCRIPTION	DATA BUS MSB LSB
Command	Write	1	Reset Command	0 0 0 0 0 0 0 0
Parameters	Write	0	Screen Comp Byte 1	S H H H H H H H
	Write	0	Screen Comp Byte 2	V V R R R R R R
	Write	0	Screen Comp Byte 3	U U U U L L L L
	Write	0	Screen Comp Byte 4	M 1 C C Z Z Z Z

Action—After the reset command is written, BRDY goes inactive, 8276H interrupts are disabled, and the VSP output is used to blank the screen. HRTC and VRTC continue to run. HRTC and VRTC timing are random on power-up.

As parameters are written, the screen composition is defined.

Parameter—S Spaced Rows

S	FUNCTIONS
0	Normal Rows
1	Spaced Rows

Parameter—HHHHHHH Horizontal Characters/Row

H H H H H H H	NO. OF CHARACTERS PER ROW
0 0 0 0 0 0 0	1
0 0 0 0 0 0 1	2
0 0 0 0 0 1 0	3
.	.
.	.
.	.
1 0 0 1 1 1 1	80
1 0 1 0 0 0 0	Undefined
.	.
.	.
.	.
1 1 1 1 1 1 1	Undefined

Parameter—VV Vertical Retrace Row Count

V V	NO. OF ROW COUNTS PER VRTC
0 0	1
0 1	2
1 0	3
1 1	4

Parameter—RRRRRR Vertical Rows/Frame

R R R R R R	NO. OF ROWS/FRAME
0 0 0 0 0 0	1
0 0 0 0 0 1	2
0 0 0 0 1 0	3
.	.
.	.
.	.
1 1 1 1 1 1	64

Parameter—UUUU Underline Placement

U U U U	LINE NUMBER OF UNDERLINE
0 0 0 0	1
0 0 0 1	2
0 0 1 0	3
.	.
.	.
.	.
1 1 1 1	16

Parameter—LLLL Number of Lines per Character Row

L L L L	NO. OF LINES/ROW
0 0 0 0	1
0 0 0 1	2
0 0 1 0	3
.	.
.	.
.	.
1 1 1 1	16

Parameter—M Line Counter Mode

M	LINE COUNTER MODE
0	Mode 0 (Non-Offset)
1	Mode 1 (Offset by 1 Count)

Parameter—CC Cursor Format

C C	CURSOR FORMAT
0 0	Blinking reverse video block
0 1	Blinking underline
1 0	Non-blinking reverse video block
1 1	Non-blinking underline

ORDER NUMBER: 210666-002

Parameter—ZZZZ Horizontal Retrace Count

Z Z Z Z	NO. OF CHARACTER COUNTS PER HRTC
0 0 0 0	2
0 0 0 1	4
0 0 1 0	6
.	.
.	.
.	.
1 1 1 1	32

Note: uuuu MSB determines blanking of top and bottom lines (1 = blanked, 0 = not blanked).

2. START DISPLAY COMMAND

	OPERATION	C/P	DESCRIPTION	DATA BUS MSB LSB
Command	Write	1	Start Display	0 0 1 0 0 0 0 0
No parameters				

Action—8276H interrupts are enabled, BRDY goes active, video is enabled, Interrupt Enable and Video Enable status flags are set.

3. STOP DISPLAY COMMAND

	OPERATION	C/P	DESCRIPTION	DATA BUS MSB LSB
Command	Write	1	Stop Display	0 1 0 0 0 0 0 0
No parameters				

Action—Disables video, interrupts remain enabled, HRTC and VRTC continue to run, Video Enable status flag is reset, and the "Start Display" command must be given to reenable the display.

4. LOAD CURSOR POSITION

	OPERATION	C/P	DESCRIPTION	DATA BUS MSB LSB
Command	Write	1	Load Cursor	1 0 0 0 0 0 0 0
Parameters	Write Write	0 0	Char. Number Row Number	(Char. Position in Row) (Row Number)

Action—The 8276H is conditioned to place the next two parameter bytes into the cursor position registers. Status flag not affected.

5. ENABLE INTERRUPT COMMAND

	OPERATION	C/P	DESCRIPTION	DATA BUS MSB LSB
Command	Write	1	Enable Interrupt	1 0 1 0 0 0 0 0
No parameters				

Action—The interrupt enable flag is set and interrupts are enabled.

6. DISABLE INTERRUPT COMMAND

	OPERATION	C/P	DESCRIPTION	DATA BUS MSB LSB
Command	Write	1	Disable Interrupt	1 1 0 0 0 0 0 0
No parameters				

Action—Interrupts are disabled and the interrupt enable status flag is reset.

7. PRESET COUNTERS COMMAND

	OPERATION	C/P	DESCRIPTION	DATA BUS MSB LSB
Command	Write	1	Preset Counters	1 1 1 0 0 0 0 0
No parameters				

Action—The internal timing counters are preset, corresponding to a screen display position at the top left corner. Two character clocks are required for this operation. The counters will remain in this state until any other command is given.

This command is useful for system debug and synchronization of clustered CRT displays on a single CPU. After this command, two additional clock cycles are required before the first character of the first row is put out.

Status Flags

	OPERATION	C/P	DESCRIPTION	DATA BUS MSB LSB
Command	Read	1	Status Word	0 IE IR X IC VE BU X

IE — (Interrupt Enable) Set or reset by command. It enables vertical retrace interrupt. It is automatically set by a "Start Display" command and reset with the "Reset" command.

IR — (Interrupt Request) This flag is set at the beginning of display of the last row of the frame if the interrupt enable flag is set. It is reset after a status read operation.

IC — (Improper Command) This flag is set when a command parameter string is too long or too short. The flag is automatically reset after a status read.

VE — (Video Enable) This flag indicates that video operation of the CRT is enabled. This flag is set on a "Start Display" command, and reset on a "Stop Display" or "Reset" command.

BU — (Buffer Underrun) This flag is set whenever a Row Buffer is not filled with character data in time for a buffer swap required by the display. Upon activation of this bit, buffer loading ceases, and the screen is blanked until after the vertical retrace interval.

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias .... 0°C to 70°C
Storage Temperature ..... −65°C to +150°C
Voltage On Any Pin
With Respect to Ground .....−0.5V to +7V
Power Dissipation ..... 1 Watt

*NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied.

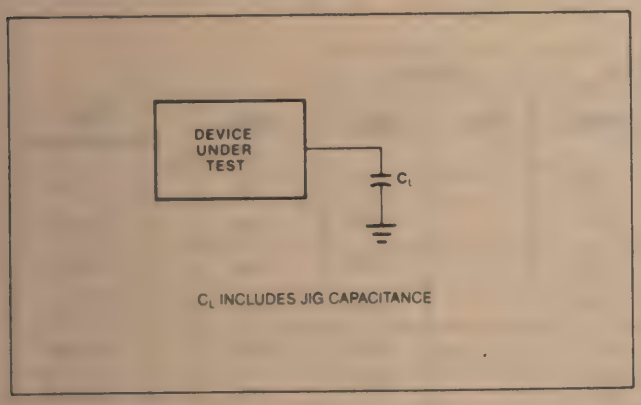
D.C. CHARACTERISTICS (T_A = 0°C to 70°C; V_{CC} = 5V ±5%)

SYMBOL	PARAMETER	MIN.	MAX.	UNITS	TEST CONDITIONS
V_{IL}	Input Low Voltage	−0.5	0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC} + 0.5V	V	
V_{OL}	Output Low Voltage		0.45	V	I_{OL} = 2.2 mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = −400 μA
I_{IL}	Input Load Current		±10	μA	V_{IN} = V_{CC} to 0V
I_{OFL}	Output Float Leakage		±10	μA	V_{OUT} = V_{CC} to 0.45V
I_{CC}	V_{CC} Supply Current		160	mA	

CAPACITANCE (T_A = 25°C; V_{CC} = GND = 0V)

SYMBOL	PARAMETER	MIN.	MAX.	UNITS	TEST CONDITIONS
C_{IN}	Input Capacitance		10	pF	f_C = 1 MHz
$C_{I/O}$	I/O Capacitance		20	pF	Unmeasured pins returned to V_{SS}.

A.C. TESTING LOAD CIRCUIT



ORDER NUMBER: 210668-002

A.C. CHARACTERISTICS (T_A = 0°C to 70°C; V_{CC} = 5.0V ±5%; GND = 0V)

Bus Parameters

READ CYCLE

Symbol	Parameter	Min.	Max.	Units	Test Conditions
t_{AR}	Address Stable Before READ	0		ns	
t_{RA}	Address Hold Time for READ	0		ns	
t_{RR}	READ Pulse Width	250		ns	
t_{RD}	Data Delay from READ		200	ns	C_L = 150pF
t_{DF}	READ to Data Floating		100	ns	

WRITE CYCLE

Symbol	Parameter	Min.	Max.	Units	Test Conditions
t_{AW}	Address Stable Before WRITE	0		ns	
t_{WA}	Address Hold Time for WRITE	0		ns	
t_{WW}	WRITE Pulse Width	250		ns	
t_{DW}	Data Setup Time for WRITE	150		ns	
t_{WD}	Data Hold Time for WRITE	0		ns	

CLOCK TIMING

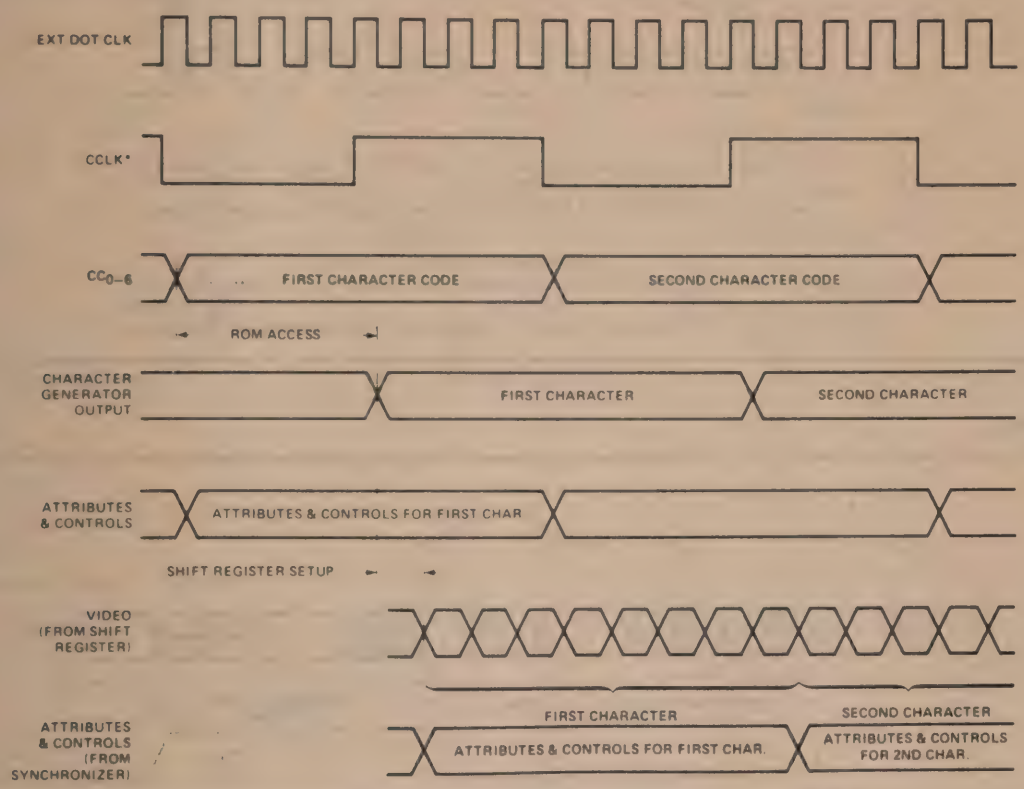
		8276H		8276-2			
Symbol	Parameter	Min.	Max.	Min.	Max.	Units	Test Conditions
t_{CLK}	Clock Period	480		320		ns	
t_{KH}	Clock High	240		120		ns	
t_{KL}	Clock Low	160		120		ns	
t_{KR}	Clock Rise	5	30	5	30	ns	
t_{KF}	Clock Fall	5	30	5	30	ns	

OTHER TIMING

		8276H		8276-2			
Symbol	Parameter	Min.	Max.	Min.	Max.	Units	Test Conditions
t_{CC}	Character Code Output Delay		150		150	ns	CL = 50 pF
t_{HR}	Horizontal Retrace Output Delay		200		150	ns	C_L = 50 pF
t_{LC}	Line Count Output Delay		400		250	ns	C_L = 50 pF
t_{AT}	Control/Attribute Output Delay		275		250	ns	C_L = 50 pF
t_{VR}	Vertical Retrace Output Delay		275		250	ns	C_L = 50 pF
t_{RI}	INT↓ from $\overline{RD}$↑		250		250	ns	C_L = 50 pF
t_{WQ}	BRDY↑ from $\overline{WR}$↑		250		250	ns	C_L = 50 pF
t_{RQ}	BRDY↓ from $\overline{WR}$↓		200		200	ns	C_L = 50 pF
t_{LR}	$\overline{BS}$↓ to $\overline{WR}$↓	0		0		ns	
t_{RL}	$\overline{WR}$↑ to $\overline{BS}$↑	0		0		ns	

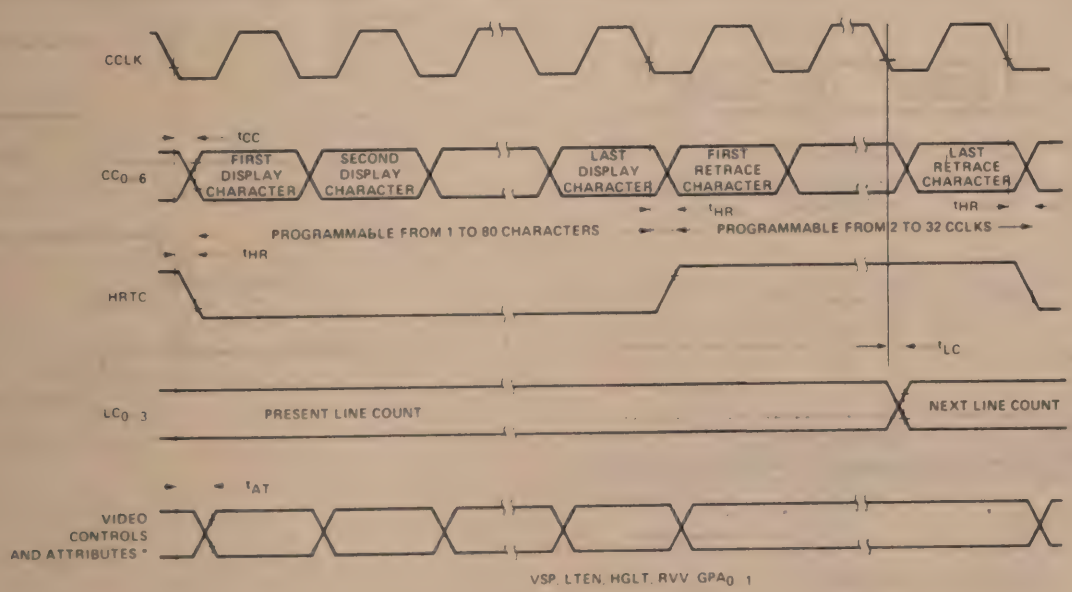
WAVEFORMS

Typical Dot Level Timing



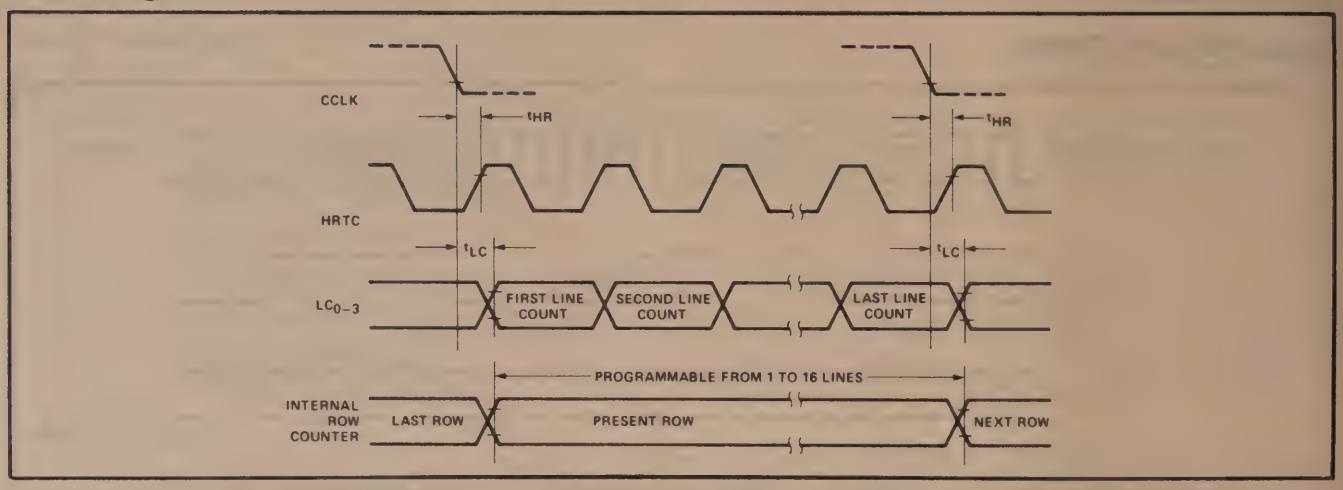
*CCLK IS A MULTIPLE OF THE DOT CLOCK AND AN INPUT TO THE 8276.

Line Timing

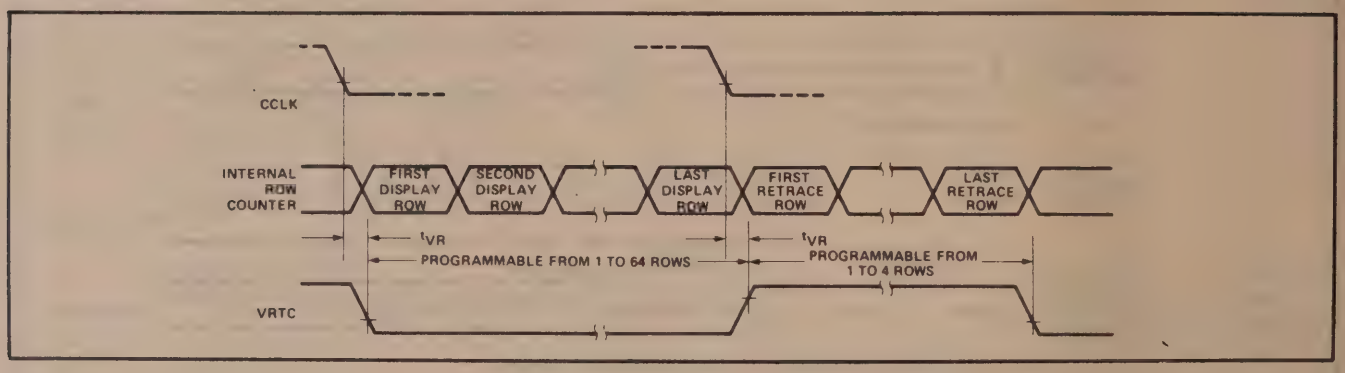


ORDER NUMBER: 210668-002

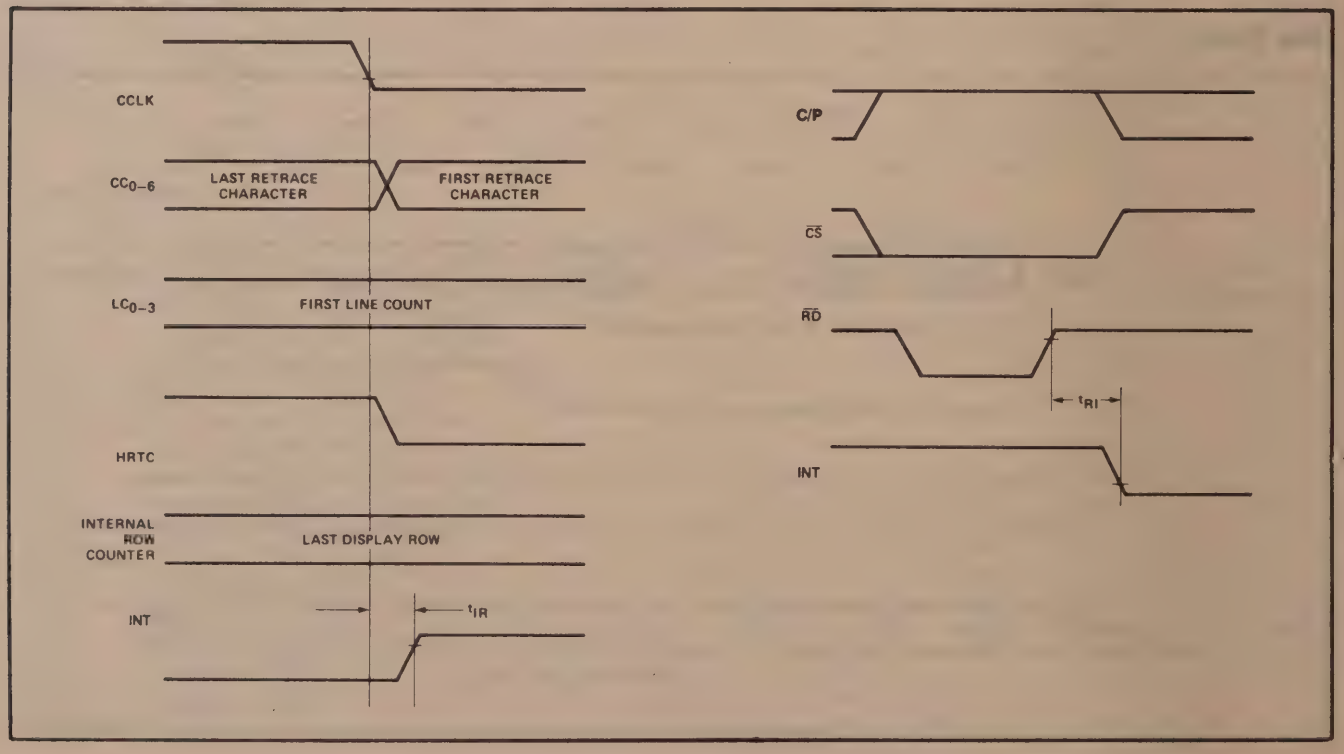
Row Timing



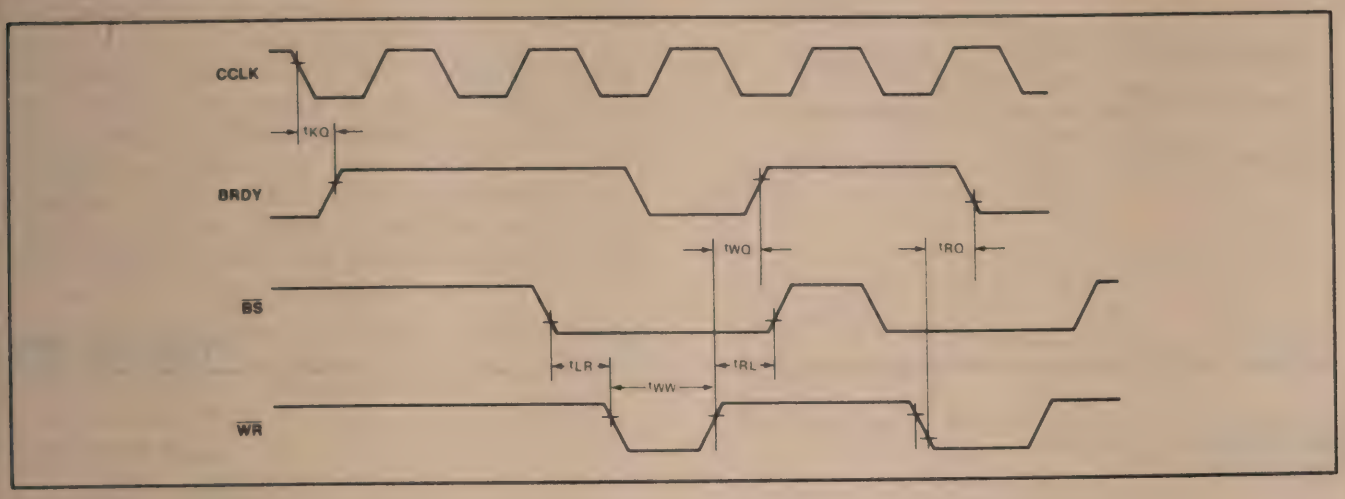
Frame Timing



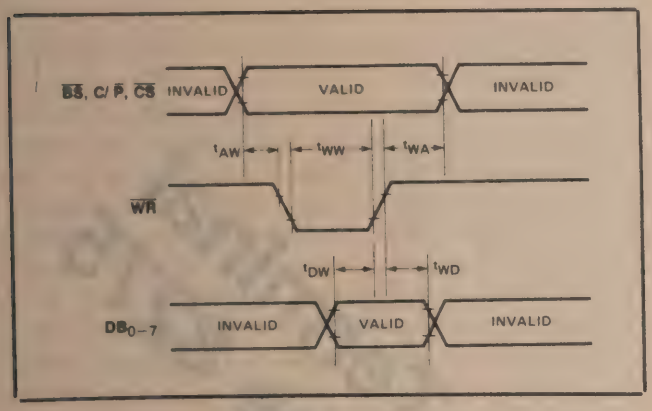
Interrupt Timing



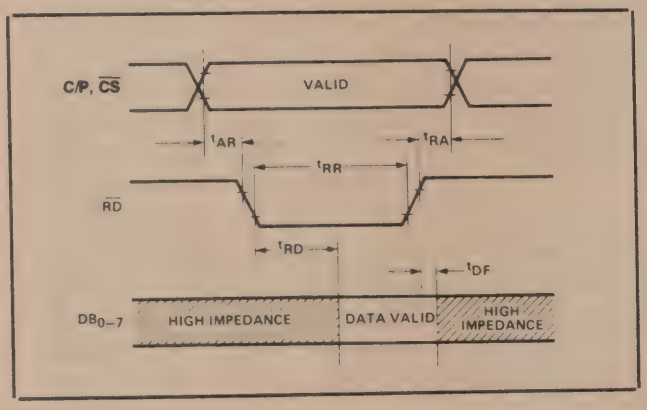
Timing for Buffer Loading



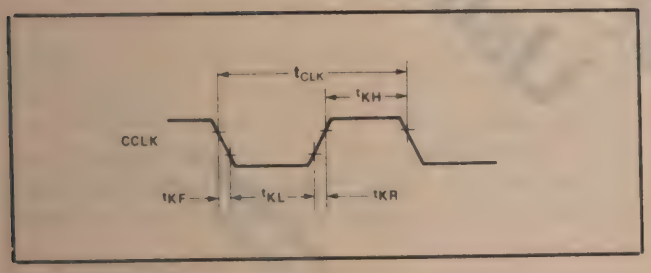
Write Timing



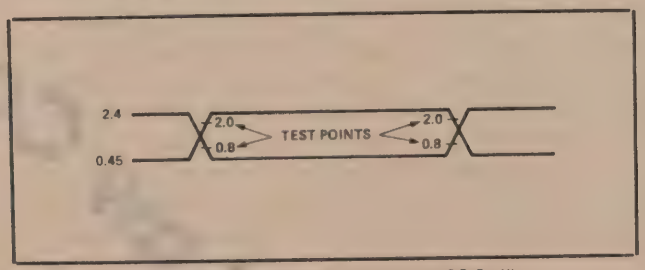
Read Timing



Clock Timing



Input and Output Waveforms for A.C. Tests



FOR A.C. TESTING, INPUTS ARE DRIVEN AT 2.4V FOR A LOGIC "1" AND 0.45V FOR A LOGIC "0." TIMING MEASUREMENTS FOR INPUT AND OUTPUT SIGNALS ARE MADE AT 2.0V FOR A LOGIC "1" AND 0.8V FOR A LOGIC "0."

ORDER NUMBER: 210669-002

intel

APPLICATION NOTE

AP-62

November 1979

A Low Cost CRT Terminal Using The 8275

John Katausky
Peripherals Applications

© Intel Corporation, 1979

207780-001

1. INTRODUCTION

The purpose of this application note is to provide the reader with the design concepts and factual tools needed to integrate Intel peripherals and microprocessors into a low cost raster scan CRT terminal. A previously published application note, AP-32, presented one possible solution to the CRT design question. This application note expands upon the theme established in AP-32 and demonstrates how to design a functional CRT terminal while keeping the parts count to a minimum.

For convenience, this application note is divided into seven general sections:

1. Introduction
2. CRT Basics
3. 8275 Description
4. Design Background
5. Circuit Description
6. Software Description
7. Appendix

There is no question that microprocessors and LSI peripherals have had a significant role in the evolution of CRT terminals. Microprocessors have allowed design engineers to incorporate an abundance of sophisticated features into terminals that were previously mere slaves to a larger processor. To complement microprocessors, LSI peripherals have reduced component count in many support areas. A typical LSI peripheral easily replaces between 30 and 70 SSI and MSI packages, and offers features and flexibility that are usually not available in most hardware designs. In addition to replacing a whole circuit board of random logic, LSI circuits also reduce the cost and increase the reliability of design. Fewer interconnects increases mechanical reliability and fewer parts decreases the power consumption and hence, the overall reliability of the design. The reduction of components also yields a circuit that is easier to debug during the actual manufacturing phase of a product.

Until the era of advanced LSI circuitry, a typical CRT terminal consisted of 80 to 200 or more SSI and MSI packages. The first microprocessors and peripherals dropped this component count to between 30 and 50 packages. This application note describes a CRT terminal that uses 20 packages.

2. CRT BASICS

The raster scan display gets its name from the fact that the image displayed on the CRT is built up by generating a series of lines (raster) across the face of the CRT. Usually, the beam starts in the upper left hand corner of the display and simultaneously moves left to right and top to bottom to put a series

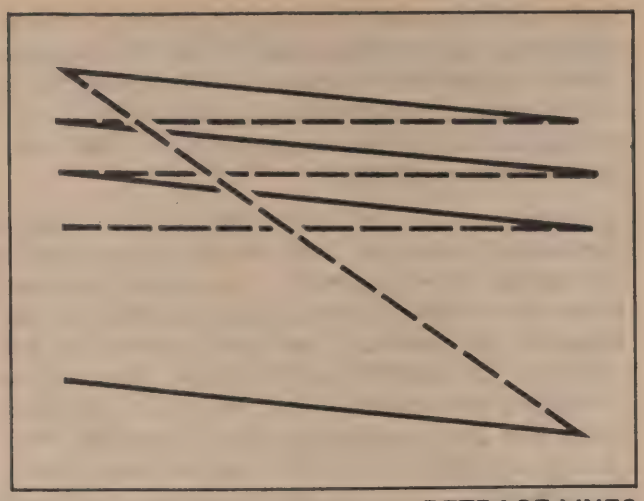


Figure 2-1. Raster Scan

of zig-zag lines on the screen (Fig. 2.1). Two simultaneously operating independent circuits control the vertical and horizontal movement of the beam.

As the electron beam moves across the face of the CRT, a third circuit controls the current flowing in the beam. By varying the current in the electron beam the image on the CRT can be made to be as bright or as dark as the user desires. This allows any desired pattern to be displayed.

When the beam reaches the end of a line, it is brought back to the beginning of the next line at a rate that is much faster than was used to generate the line. This action is referred to as "retrace". During the retrace period the electron beam is usually shut off so that it doesn't appear on the screen.

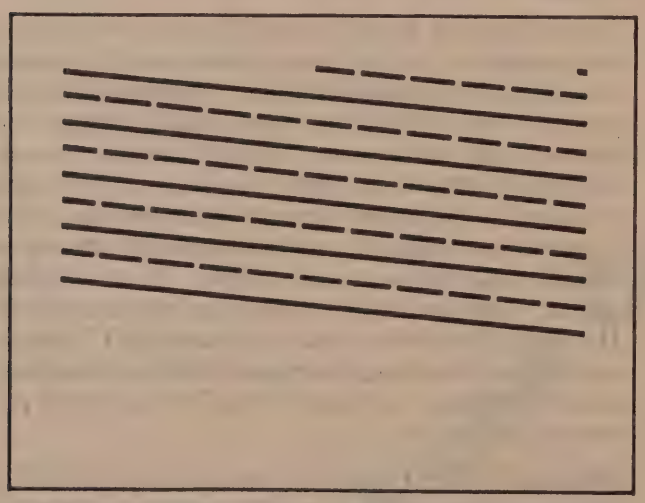
As the electron beam is moving across the screen horizontally, it is also moving downward. Because of this, each successive line starts slightly below the previous line. When the beam finally reaches the bottom right hand corner of the screen, it retraces vertically back to the top left hand corner. The time it takes for the beam to move from the top of the screen to the bottom and back again to the top is usually referred to as a "frame". In the United States, commercial television broadcast use 15,750 Hz as the horizontal sweep frequency (63.5 microseconds per horizontal line) and 60 Hz as the vertical sweep frequency or "frame" (16.67 milliseconds per vertical frame).

Although, the 60 Hz vertical frame and the 15,750 Hz horizontal line are the standards used by commercial broadcasts, they are by no means the only frequency at which CRT's can operate. In fact, many CRT displays use a horizontal scan that is around 18 KHz to 22 KHz and some even exceed 30 KHz. As the

horizontal frequency increases, the number of horizontal lines per frame increases. Hence, the resolution on the vertical axis increases. This increased resolution is needed on high density graphic displays and on special text editing terminals that display many lines of text on the CRT.

Although many CRT's operate at non-standard horizontal frequencies, very few operate at vertical frequencies other than 60 Hz. If a vertical frequency other than 60 Hz is chosen, any external or internal magnetic or electrical variations at 60 Hz will modulate the electron beam and the image on the screen will be unstable. Since, in the United States, the power line frequency happens to be 60 Hz, there is a good chance for 60 Hz interference to exist. Transformers can cause 60 Hz magnetic fields and power supply ripple can cause 60 Hz electrical variations. To overcome this, special shielding and power supply regulation must be employed. In this design, we will assume a standard frame rate of 60 Hz and a standard line rate of 15,750 Hz.

By dividing the 63.5 microsecond horizontal line rate into the 16.67 millisecond vertical rate, it is found that there are 262.5 horizontal lines per vertical frame. At first, the half line may seem a bit odd, but actually it allows the resolution on the CRT to be effectively doubled. This is done by inserting a second set of horizontal lines between the first set (interlacing). In an interlaced system the line sets are not generated simultaneously. In a 60 Hz system, first all of the even-numbered lines are scanned: 0, 2, 4,...524. Then all the odd-numbered lines: 1, 3, 5,... 525. Each set of lines usually contains different data (Fig. 2.2).



EVEN FIELD
ODD FIELD

RETRACE LINES NOT SHOWN

Figure 2-2. Interlaced Scan

Although interlacing provides greater resolution, it also has some distinct disadvantages. First of all, the circuitry needed to generate the extra half horizontal line per frame is quite complex when compared to a noninterlaced design, which requires an integer number of horizontal lines per frame. Next, the overall vertical refresh rate is half that of a noninterlaced display. As a result, flicker may result when the CRT uses high speed phosphors. To keep things as simple as possible, this design uses the noninterlaced approach.

The first thing any CRT controller must do is generate pulses that define the horizontal line timing and the vertical frame timing. This is usually done by dividing a crystal reference source by some appropriate numbers. On most raster scan CRT's the horizontal frequency is very forgiving and can vary by around 500 Hz or so and produce no ill effects. This means that the CRT itself can track a horizontal frequency between 15250 Hz and 16250 Hz, or in other words, there can be 256 to 270 horizontal lines per vertical frame. But, as mentioned earlier, the vertical frequency should be 60 Hz to insure stability.

The characters that are viewed on the screen are formed by a series of dots that are shifted out of the controller while the electron beam moves across the face of the CRT. The circuits that create this timing are referred to as the dot clock and character clock. The character clock is equal to the dot clock divided by the number of dots used to form a character along the horizontal axis and the dot clock is calculated by the following equation:

$$\text{DOT CLOCK (Hz)} = (N + R) * D * L * F$$

where N is the number of displayed characters per row,

R is the number of retrace character time increments,

D is the number of dots per character,

L is the number of horizontal lines per frame and

F is the frame rate in Hz.

In this design N = 80, R = 20, D = 7, L = 270, and F = 60 Hz. If the numbers are plugged in, the dot clock is found to be 11.34 MHz.

The retrace number, R, may vary from system to system because it is used to establish the margins on the left and right hand sides of the CRT. In this particular design R = 20 was empirically found it be optimum. The number of dots per character may vary depending on the character generator used and the number of dot clocks the designer wants to place between characters. This design uses a 5 X 7 dot matrix and allows 2 dot clock periods between characters (see Fig. 2.3); since 5 + 2 equals 7, we find that D = 7.

Figure 2-3. 5 X 7 Dot Matrix

The number of lines per frame can be determined by the following equation:

$$L = (H * Z) + V$$

where, H is the number of horizontal lines per character,

Z is the number of character lines per frame and

V is the number of horizontal lines during vertical retrace. In this design, a 5 X 7 dot matrix is to be placed on a 7 X 10 field, so H = 10. Also, 25 lines are to be displayed, so Z = 25. As mentioned before, V = 20. When the numbers are plugged into the equation, L is found to be equal to 270 lines per frame.

The designer should be cautioned that these numbers are interrelated and that to guarantee proper operation on a standard raster scan CRT, L should be between 256 and 270. If L does not lie within these bounds the horizontal circuits of the CRT may not be able to lock onto the driving signal and the image will roll horizontally. The chosen L of 270 yields a horizontal frequency of 16,200 KHz on a 60 Hz frame and this number is within the 500 Hz tolerance mentioned earlier.

The V number is chosen to match the CRT in much the same manner as the R number mentioned earlier. When the electron beam reaches the bottom right corner of the screen it must retrace vertically to the top left corner. This retrace action requires time, usually between 900-1200 microseconds. To allow for this, enough horizontal sync times must be inserted during vertical retrace. Twenty horizontal sync times at 61.5 microseconds yield a total of 1234.5 microseconds, which is enough time to allow the beam to return to the top of the screen.

The choices of H and Z largely relate to system design preference. As H increases, the character size along the vertical axis increases. Z is simply the number of lines of characters that are displayed and this, of course, is entirely a system design option.

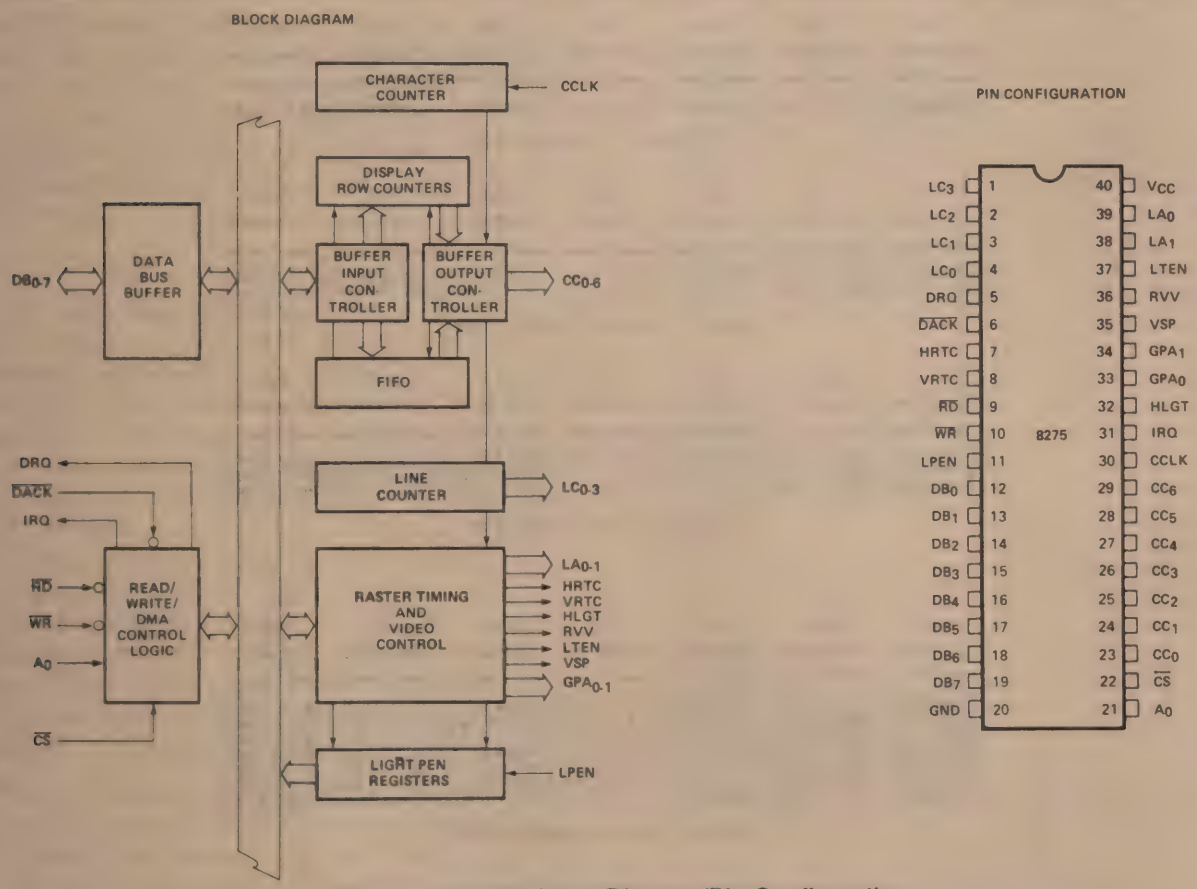


Figure 3-1. 8275 Block Diagram/Pin Configuration

3. 8275 DESCRIPTION

A block diagram and pin configuration of the 8275 are shown in Fig. 3.1. The following is a description of the general capabilities of the 8275.

3.1 CRT DISPLAY REFRESHING

The 8275, having been programmed by the designer to a specific screen format, generates a series of DMA request signals, resulting in the transfer of a row of characters from display memory to the 8275's row buffers. The 8275 presents the character codes to an external character generator ROM by using outputs CC0-CC6. External dot timing logic is then used to transfer the parallel output data from the character generator ROM serially to the video input of the CRT. The character rows are displayed on the CRT one line at a time. Line count outputs LC0-LC3 are applied to the character generator ROM to perform the line selection function. The display process is illustrated in Figure 3.2. The entire process is repeated for each display row. At the beginning of the last displayed row, the 8275 issues an interrupt by setting the IRQ output line. The 8275 interrupt output will normally be connected to the interrupt input of the system central processor. The interrupt causes the CPU to execute an interrupt service subroutine. The service subroutine typically re-initializes DMA controller parameters for the next display refresh cycle, polls the system keyboard controller, and/or executes other appropriate functions. A block diagram of a CRT system implemented with the 8275 CRT Controller is provided in Figure 3.3. Proper CRT refreshing requires that certain 8275 parameters be programmed prior to the beginning of display operation. The 8275 has two types of programming registers, the Command Registers (CREG) and the Parameter Registers (PREG). It also has a Status Register (SREG). The Command Registers may only be written to and the Status Registers may only be read. The 8275 expects to receive a command followed by a sequence of from 0 to 4 parameters, depending on the command. The 8275 instruction set consist of the eight commands shown in Figure 3.4.

To establish the format of the display, the 8275 provides a number of user programmable display format parameters. Display formats having from 1 to 80 characters per row, 1 to 64 rows per screen, and 1 to 16 horizontal lines per row are available.

In addition to transferring characters from memory

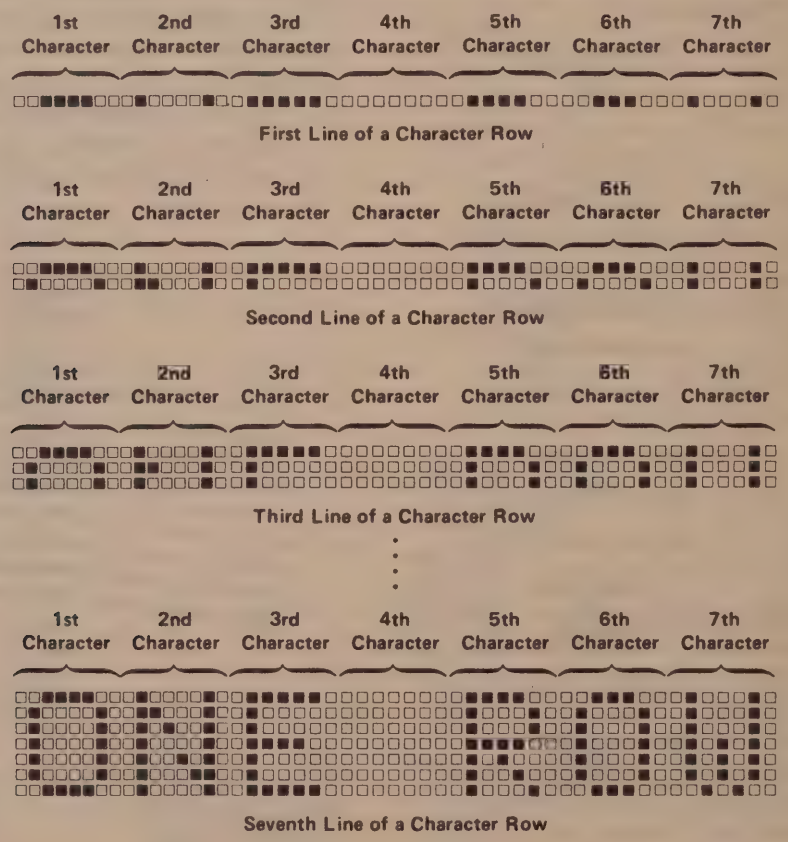


Figure 3-2. 8275 Row Display

207780-001

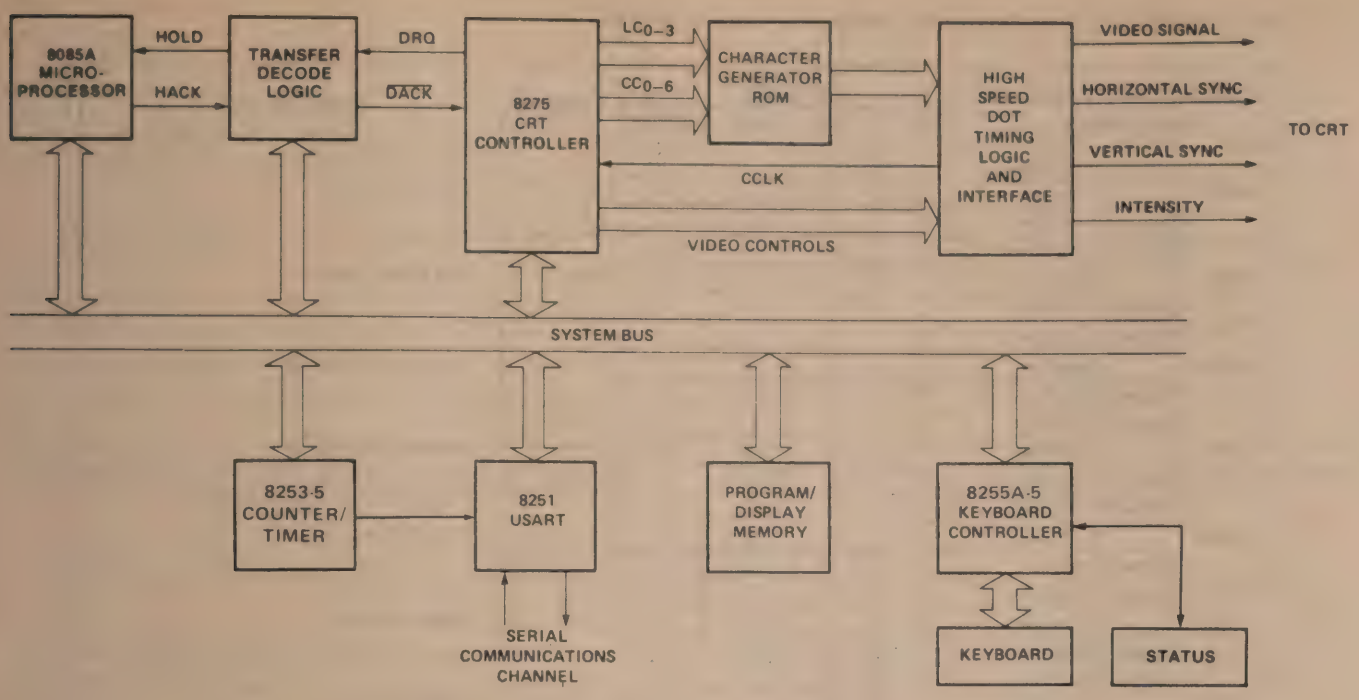


Figure 3-3. CRT System Block Diagram

to the CRT screen, the 8275 features cursor position control. The cursor position may be programmed, via X and Y cursor position registers, to any character position on the display. The user may select from four cursor formats. Blinking or non-blinking underline and reverse video block cursors are available.

3.2 CRT TIMING

The 8275 provides two timing outputs, HRTC and VRTC, which are utilized in synchronizing CRT horizontal and vertical oscillators to the 8275 refresh cycle. In addition, whenever HRTC or VRTC is active, a third timing output, VSP (Video Suppress) is true, providing a blinking signal to the dot timing logic. The dot timing logic will normally inhibit the video output to the CRT during the time when video suppress signal is true. An additional timing output, LTEN (Light Enable) is used to provide the ability to force the video output high regardless of the state of VSP. This feature is used by the 8275 to place a cursor on the screen and to control attribute functions. Attributes will be considered in the next section.

The HLGT (Highlight) output allows an attribute function to increase the CRT beam intensity to a level greater than normal. The fifth timing signal, RVV (Reverse Video) will, when enabled, cause the system video output to be inverted.

COMMAND	NO. OF PARAMETER BYTES	NOTES
RESET	4	Display format parameters required
START DISPLAY	0	DMA operation parameters included in command
STOP DISPLAY	0	—
READ LIGHT PEN	2	—
LOAD CURSOR	2	Cursor X,Y position parameters required
ENABLE INTERRUPT	0	—
DISABLE INTERRUPT	0	—
PRESET COUNTERS	0	Clears all internal counters

Figure 3-4. 8275's Instruction Set

Character attributes were designed to produce the following graphics:

CHARACTER ATTRIBUTE CODE "CCCC"		OUTPUTS				SYMBOL	DESCRIPTION
		LA_1	LA_0	VSP	LTEN		
0000	Above Underline	0	0	1	0		Top Left Corner
	Underline	1	0	0	0		
	Below Underline	0	1	0	0		
0001	Above Underline	0	0	1	0		Top Right Corner
	Underline	1	1	0	0		
	Below Underline	0	1	0	0		
0010	Above Underline	0	1	0	0		Bottom Left Corner
	Underline	1	0	0	0		
	Below Underline	0	0	1	0		
0011	Above Underline	0	1	0	0		Bottom Right Corner
	Underline	1	1	0	0		
	Below Underline	0	0	1	0		
0100	Above Underline	0	0	1	0		Top Intersect
	Underline	0	0	0	1		
	Below Underline	0	1	0	0		
0101	Above Underline	0	1	0	0		Right Intersect
	Underline	1	1	0	0		
	Below Underline	0	1	0	0		
0110	Above Underline	0	1	0	0		Left Intersect
	Underline	1	0	0	0		
	Below Underline	0	1	0	0		
0111	Above Underline	0	1	0	0		Bottom Intersect
	Underline	0	0	0	1		
	Below Underline	0	0	1	0		
1000	Above Underline	0	0	1	0		Horizontal Line
	Underline	0	0	0	1		
	Below Underline	0	0	1	0		
1001	Above Underline	0	1	0	0		Vertical Line
	Underline	0	1	0	0		
	Below Underline	0	1	0	0		
1010	Above Underline	0	1	0	0		Crossed Lines
	Underline	0	0	0	1		
	Below Underline	0	1	0	0		
1011	Above Underline	0	0	0	0		Not Recommended *
	Underline	0	0	0	0		
	Below Underline	0	0	0	0		
1100	Above Underline	0	0	1	0		Special Codes
	Underline	0	0	1	0		
	Below Underline	0	0	1	0		
1101	Above Underline						Illegal
	Underline		Undefined				
	Below Underline						
1110	Above Underline						Illegal
	Underline		Undefined				
	Below Underline						
1111	Above Underline						Illegal
	Underline		Undefined				
	Below Underline						

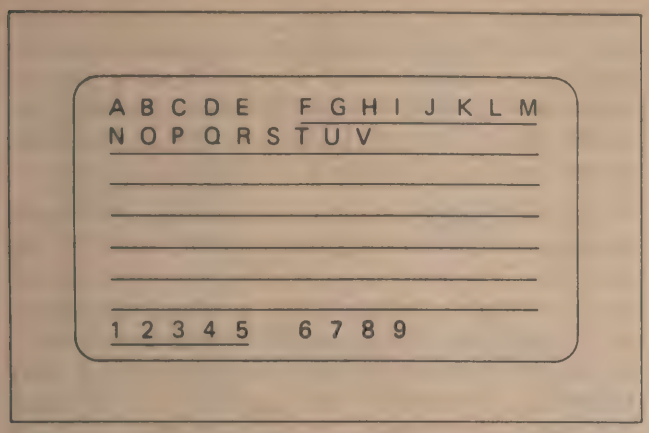
*Character Attribute Code 1011 is not recommended for normal operation. Since none of the attribute outputs are active, the character Generator will not be disabled, and an indeterminate character will be generated.

Character Attribute Codes 1101, 1110, and 1111 are illegal.

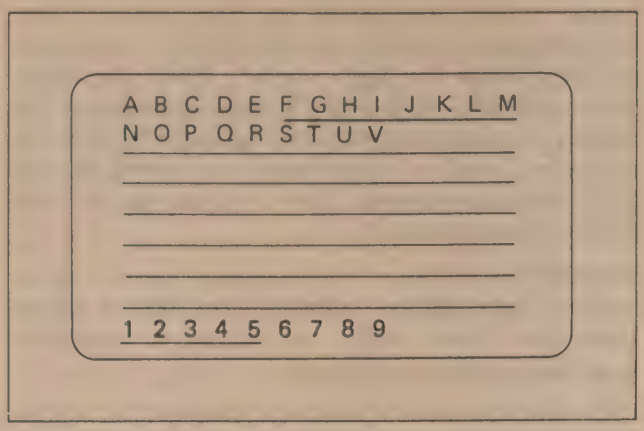
Blinking is active when B = 1.

Highlight is active when H = 1.

Figure 3-5. Character Attributes



EXAMPLE OF THE VISIBLE FIELD ATTRIBUTE MODE (UNDERLINE ATTRIBUTE)



EXAMPLE OF THE INVISIBLE FIELD ATTRIBUTE MODE (UNDERLINE ATTRIBUTE)

Figure 3-6. Field Attribute Examples

3.3 SPECIAL FUNCTIONS

VISUAL ATTRIBUTES—Visual attributes are special codes which, when retrieved from display memory by the 8275, affect the visual characteristics of a character position or field of characters. Two types of visual attributes exist, character attributes and field attributes.

Character Attribute Codes: Character attribute codes can be used to generate graphics symbols without the use of a character generator. This is accomplished by selectively activating the Line Attribute outputs (LAO-LA1), the Video Suppression output (VSP), and the Light Enable output (LTEN). The dot timing logic uses these signals to generate the proper symbols. Character attributes can be programmed to blink or be highlighted individually. Blinking is accomplished with the Video Suppression output (VSP). Blink frequency is equal to the screen refresh frequency divided by 32. Highlighting is accomplished by activating the Highlight output (HGLT). Character attributes were designed to produce the graphic symbols shown in Figure 3.5.

Field Attribute Codes: The field attributes are control codes which affect the visual characteristics for a field of characters, starting at the character following the field attribute code up to, and including, the character which precedes the next field attribute code, or up to the end of the frame.

There are six field attributes:

1. *Blink* — Characters following the code are caused to blink by activating the Video Suppression output (VSP). The blink frequency is equal to the screen refresh frequency divided by 32.
2. *Highlight* — Characters following the code are caused to be highlighted by activating the Highlight output (HGLT).
3. *Reverse Video* — Characters following the code are caused to appear in reverse video format by activating the Reverse Video output (RVV).
4. *Underline* — Characters following the code are caused to be underlined by activating the Light Enable output (LTEN).
5. *General Purpose* — There are two additional 8275 outputs which act as general purpose, independently programmable field attributes. These attributes may be used to select colors or perform other desired control functions.

The 8275 can be programmed to provide visible or invisible field attribute characters as shown in Figure 3.6. If the 8275 is programmed in the visible field attribute mode, all field attributes will occupy a position on the screen. They will appear as blanks caused by activation of the Video Suppression output (VSP). The chosen visual attributes are activated after this blanked character. If the 8275 is programmed in the invisible field attribute mode, the 8275 row buffer FIFOs are activated. The FIFOs effectively lengthen the row buffers by 16 characters, making room for up to 16 field attribute characters per display row. The FIFOs are 126 characters by 7 bits in size. When a field attribute is placed in the row buffer during DMA, the buffer input controller recognizes it and places the next character in the proper FIFO. When a field attribute is placed in the buffer output controller during display, it causes the controller to immediately put a character from the FIFO on the Character Code outputs (CCO-6). The chosen attributes are also activated.

LIGHT PEN DETECTION — A light pen consists fundamentally of a switch and light sensor. When the light pen is pressed against the CRT screen, the switch enables the light sensor. When the raster sweep coincides with the light sensor position on the display, the light pen output is input and the row and character position coordinates are stored in two 8275 internal registers. These registers can be read by the microprocessor.

SPECIAL CODES — Four special codes may be used to help reduce memory, software, or DMA overhead. These codes are placed in character positions in display memory.

1. *End Of Row Code* - Activates VSP. VSP remains active until the end of the line is reached. While VSP is active, the screen is blanked.
2. *End Of Row-Stop DMA Code* - Causes the DMA Control Logic to stop DMA for the rest of the row when it is written into the row buffer. It affects the display in the same way as the End of Row Code.
3. *End Of Screen Code* - Activates VSP. VSP remains active until the end of the frame is reached.
4. *End Of Screen-Stop DMA Code* - Causes the DMA Control Logic to stop DMA for the rest of the frame when it is written into the row buffer. It affects the display in the same way as the End of Screen Code.

PROGRAMMABLE DMA BURST CONTROL — The 8275 can be programmed to request single-byte DMA transfers of DMA burst transfers of 2, 4, or 8 characters per burst. The interval between bursts is also programmable. This allows the user to tailor the DMA overhead to fit the system needs.

4. DESIGN BACKGROUND

4.1 DESIGN PHILOSOPHY

Since the cost of any CRT system is somewhat proportional to parts count, arriving at a minimum part count solution without sacrificing performance has been the motivating force throughout this design effort. To successfully design a CRT terminal and keep the parts count to a minimum, a few things became immediately apparent.

1. An 8085 should be used.
2. Address and data buffering should be eliminated.
3. Multi-port memory should be eliminated.
4. DMA should be eliminated.

Decision 1 is obvious, the 8085's on-board clock generator, bus controller and vectored interrupts greatly reduce the overall part count considerably. Decision 2 is fairly obvious; if a circuit can be designed so that loading on the data and address lines is kept to a minimum, both the data and address buffers can be eliminated. This easily saves three to eight packages and reduces the power consumption of the design. Both decisions 3 and 4 require a basic understanding of current CRT design concepts.

In any CRT design, extreme time conflicts are created because all essential elements require access to the bus. The CPU needs to access the memory to control the system and to handle the incoming characters, but, at the same time, the CRT controller needs to access the memory to keep the raster scan display refreshed. To resolve this conflict two common techniques are employed, page buffering and line buffering.

In the page buffering approach the entire screen memory is isolated from the rest of the system. This isolation is usually accomplished with three-state buffers or two line to one line multiplexers. Of course, whenever a character needs to be manipulated the CPU must gain access to the buffered memory and, again, possible contention between the CPU and the CRT controller results. This contention is usually resolved in one of two ways, (1) the CPU is always given priority, or; (2) the CPU is allowed to access the buffered memory only during horizontal and vertical retrace times.

Approach 1 is the easiest to implement from a hardware point of view, but if the CPU always has priority the display may temporarily blink or "flicker" while the CPU accesses the display memory. This, of course, occurs because when the CPU accesses the display memory the CRT controller is not able to retrieve a character, so the display must be blanked during this time. Aesethically, this "flickering" is not desirable, so approach 2 is often used.

The second approach eliminates the display flickering encountered in the previously mentioned technique, but additional hardware is required. Usually the vertical and horizontal blank signals are gated with the buffered memory select lines and this line is used to control the CPU's ready line. So, if the CPU wants to use the buffered memory, its ready line is asserted until horizontal or vertical retrace times. This, of course, will impact the CPU's overall through put.

Both page buffered approaches require a significant amount of additional hardware and for the most part are not well suited for a minimum parts count type of terminal. This guides us to the line buffered approach. This approach eliminates the separate buffered memory for the display, but, at the same time, introduces a few new problems that must be solved.

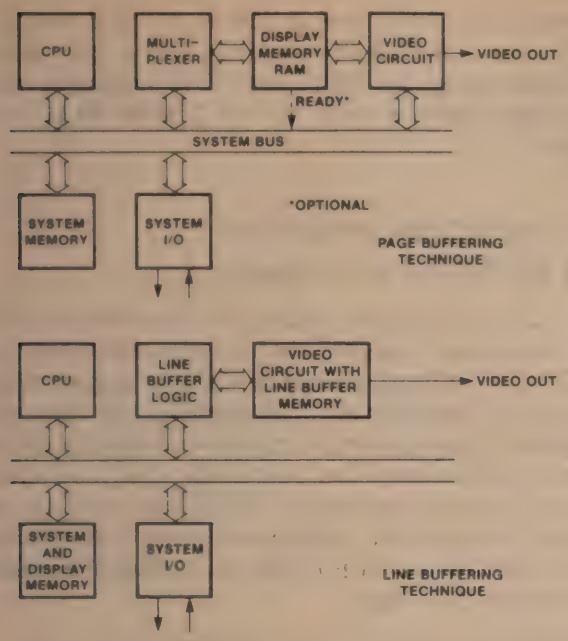


Figure 4-1. Line Buffering Technique

```
CLOCK CYCLES      SEQ        SOURCE STATEMENT

   10             1          PUSH   PSW          ;SAVE A AND FLAGS
   10             2          PUSH   H            ;SAVE H AND L
   10             3          PUSH   D            ;SAVE D AND E
   10             4          LXI    H,0000H      ;ZERO H AND L
   10             5          DAD    SP           ;PUT STACK POINTER IN H AND L
    4             6          XCHG                ;PUT STACK IN D AND E
   16             7          LHLD   CURAD        ;GET POINTER
    6             8          SPHL                ;PUT CURRENT LINE INTO SP
    7             9          MVI    A,0C0H       ;SET MASK FOR SIM
    4            10          SIM                 ;SET SPECIAL TRANSFER BIT
  400            11          POP    H            ;DO 40 POPS
    4            12          RRC                 ;SET UP A
    4            13          SIM                 ;GO BACK TO NORMAL MODE
   10            14          LXI    H,0000H      ;ZERO HL
   10            15          DAD    SP           ;ADD STACK
    4            16          XCHG                ;PUT STACK IN H AND L
    6            17          SPHL                ;RESTORE STACK
   10            18          LXI    H,LAST       ;PUT BOTTOM DISPLAY IN H AND L
    4            19          XCHG                ;SWAP REGISTERS
    4            20          MOV    A,D          ;PUT HIGH ORDER IN A
    4            21          CMP    H            ;SEE IF SAME AS H
 7/10            22          JNZ    KPTK         ;IF NOT LEAVE
    4            23          MOV    A,E          ;PUT LOW ORDER IN A
    4            24          CMP    L            ;SEE IF SAME AS L
 7/10            25          JNZ    KPTK         ;IF NOT LEAVE
   10            26          LXI    H,TPDIS      ;LOAD H AND L WITH TOP OF SCREEN MEMORY
   16            27 KPTK:    SHLD   CURAD        ;PUT BACK CURRENT ADDRESS
    7            28          MVI    A,18H        ;GET MASK BYTE
    4            29          SIM                 ;SET INTERRUPT MASK
   10            30          POP    D            ;GET D AND E
   10            31          POP    H            ;GET H AND L
   10            32          POP    PSW          ;GET A AND FLAGS
    4            33          EI                  ;ENABLE INTERRUPTS
   10            34          RET                 ;GO BACK

      TOTAL CLOCK CYCLES = 650 (WORST CASE)

      WITH A 6.144 MHZ CRYSTAL TOTAL TIME TO FILL

      ROW BUFFER ON 8275 = 650 * .325 = 211.25 MICROSECONDS
```

Figure 4-2. Routine To Load 8275's Row Buffers

In the line buffered approach both the CPU and the CRT controller share the same memory. Every time the CRT controller needs a new character or line of data, normal processing activity is halted and the CRT controller accesses memory and displays the data. Just how the CRT controller needs to acquire the display data greatly affects the performance of the overall system. Whether the CRT controller needs to gain access to the main memory to acquire a single character or a complete line of data depends on the presence or absence of a separate line or row buffer.

If no row buffer is present the CRT controller must go to the main memory to fetch every character. This of course, is not a very efficient approach because the processor will be forced to relinquish the bus 70% to 80% of the time. So much processor inactivity greatly affects the overall system performance. In fact terminals that use this approach are typically limited to around 1200 to 2400 baud on their serial communication channels. This low baud rate is in general not acceptable, hence this approach was not chosen.

If a separate row buffer is employed the CRT controller only has to access the memory once for each displayed character per line. This forces the processor to relinquish the bus only about 20% to 35% of the time and a full 4800 to 9600 baud can be achieved. Figure 4.1 illustrates these different techniques.

The 8275 CRT controller is ideal for implementing the row buffer approach because the row buffer is contained on the device itself. In fact, the 8275 contains two 80-byte row buffers. The presence of two row buffers allow one buffer to be filled while the other buffer is displaying the data. This dual row buffer approach enhances CPU performance even further.

4.2 USING THE 8275 WITHOUT DMA

Until now the process of filling the row buffer has only been alluded to. In reality, a DMA technique is usually used. This approach was demonstrated in AP-32 where an 8257 DMA controller was mated to an 8275 CRT controller. In order to minimize component count, this design eliminates the DMA controller and its associated circuitry while replacing them with a special interrupt-driven transfer.

The only real concern with using the 8275 in an interrupt-driven transfer mode is speed. Eighty characters must be loaded into the 8275 every 617 microseconds and the processor must also have time to perform all the other tasks that are required. To minimize the overhead associated with loading the characters into the 8275 a special technique was employed. This technique involves setting a special

transfer bit and executing a string of POP instructions. The string of POP instructions is used to rapidly move the data from the memory into the 8275. Figure 4.2 shows the basic software structure.

In this design the 8085's SOD line was used as the special transfer bit. In order to perform the transfer properly this special bit must do two things: (1) turn processor reads into $\overline{DACK}$ plus $\overline{WR}$ for the 8275 and (2) mask processor fetch cycles from the 8275, so that a fetch cycle does not write into the 8275. Conventional logic could have been used to implement this special function, but in this design a small bipolar programmable read only memory was used. Figure 4.3 shows a basic version of the hardware.

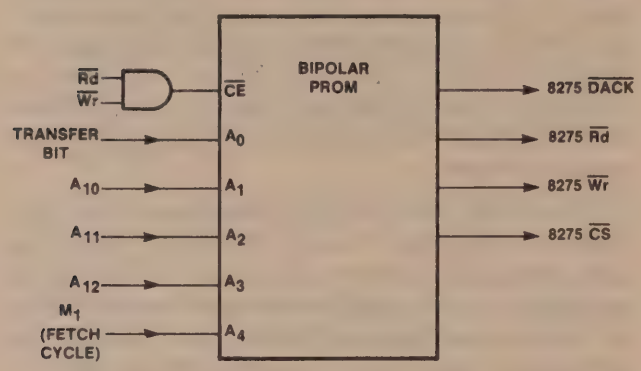


Figure 4-3. Simplified Version of Hardware Decoder

At first, it may seem strange that we are supplying a $\overline{DACK}$ when no DMA controller exist in the system. But the reader should be aware that all Intel peripheral devices that have DMA lines actually use $\overline{DACK}$ as a chip select for the data. So, when you want to write a command or read status you assert $\overline{CS}$ and $\overline{WR}$ or $\overline{RD}$, but when you want to read or write data you assert $\overline{DACK}$ and $\overline{RD}$ or $\overline{WR}$. The peripheral device doesn't "know" if a DMA controller is in the circuit or not. In passing, it should be mentioned that $\overline{DACK}$ and $\overline{CS}$ should not be asserted on the same device at the same time, since this combination yields an undefined result.

This POP technique actually compares quite favorably in terms of time to the DMA technique. One POP instruction transfers two bytes of data to the 8275 and takes 10 CPU clock cycles to execute, for a net transfer rate of one byte every five clock cycles. The DMA controller takes four clock cycles to transfer one byte but, some time is lost in synchronization. So the difference between the two techniques is one clock cycle per byte maximum. If we compare the overall speed of the 8085 to the speed of the 8080 used in AP-32, we find that at 3 MHz we can transfer one byte every 1.67 microseconds using the 8085 and POP technique vs. 2 microseconds per byte for the 2 MHz 8080 using DMA.

5. CIRCUIT DESCRIPTION

5.1 SCOPE OF THE PROJECT

A fully functional, microprocessor-based CRT terminal was designed and constructed using the 8275 CRT controller and the 8085 as the controlling element. The terminal had many of the functions found in existing commercial low-cost terminals and more sophisticated features could easily be added with a modest amount of additional software. In order to minimize component count LSI devices were used whenever possible and software was used to replace hardware.

5.2 SYSTEM TARGET SPECIFICATIONS

The design specifications for the CRT terminal were as follows:

Display Format

- 80 characters per display row
- 25 display rows

Character Format

- 5 X 7 dot matrix character contained within a 7 X 10 matrix
- First and seventh columns blanked
- Ninth line cursor position
- Blinking underline cursor

Special Characters Recognized

- Control characters
- Line feed
- Carriage Return
- Backspace
- Form feed

Escape Sequences Recognized

- ESC, A, Cursor up
- ESC, B, Cursor down
- ESC, C, Cursor right
- ESC, D, Cursor left
- ESC, E, Clear screen
- ESC, H, Home cursor
- ESC, J, Erase to the end of the screen
- ESC, K, Erase the current line

Characters Displayed

- 96 ASCII alphanumeric characters
- Special control characters

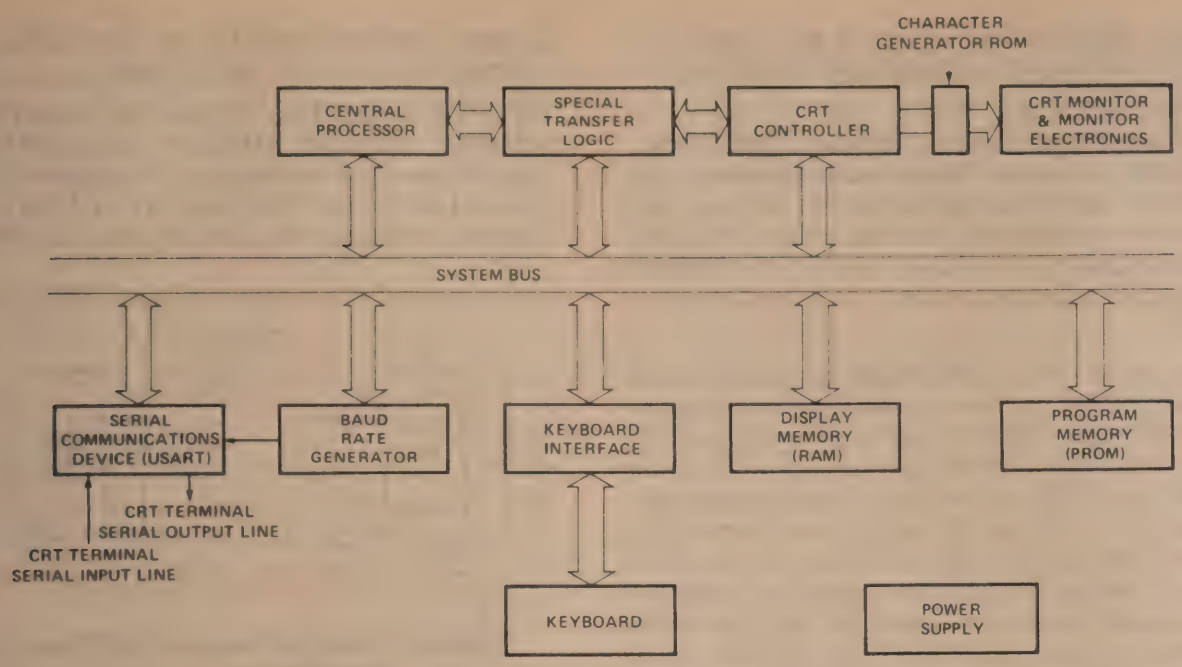


Figure 5-1. CRT Terminal Block Diagram

Characters Transmitted

- 96 ASCII alphanumeric characters
- ASCII control characters

Program Memory

- 2K bytes of 2716 EPROM

Display/Buffer/Stack Memory

- 2K bytes 2114 static memory (4 packages)

Data Rate

- 9600 BAUD using 3MHz 8085

CRT Monitor

- Ball Bros TV-12, 12MHz B.W.

Keyboard

- Any standard un-encoded ASCII keyboard

Screen Refresh Rate

- 60 Hz

5.3 HARDWARE DISCRIPTION

A block diagram of the CRT terminal is shown in Figure 5.1. The diagram shows only the essential system features. A detailed schematic of the CRT is contained in the Appendix. The terminal was constructed on a simple 6" by 6" wire wrap board. Because of the minimum bus loading no buffering of any kind was needed (see Figure 5.2).

The "heart" of the CRT terminal is the 8085 microprocessor. The 8085 initializes all devices in the system, loads the CRT controller, scans the keyboard, assembles the characters to be transmitted, decodes the incoming characters and determines where the character is to be placed on the screen. Clearly, the processor is quite busy.

A standard list of LSI peripheral devices surround the 8085. The 8251A is used as the serial communication link, the 8255A-5 is used to scan the keyboard and read the system variables through a set of

Worst case bus loading:

Data Bus:	8275	20pf
	8255A-5	20pf
	8253-5	20pf
	8253-5	20pf
	8251A	20pf
	2x 2114	10pf
	2716	12pf
	8212	12pf
		114pf max

Only A_8 - A_{15} are important since A_0 - A_7 are latched by the 8212

Address Bus:	4x 2114	20pf
	2716	6pf
		26pf max

This loading assures that all components will be compatible with a 3MHz 8085 and that no wait states will be required

Figure 5-2. Bus Loading

switches, and the 8253 is used as a baud rate generator and as a "horizontal pulse extender" for the 8275.

The 8275 is used as the CRT controller in the system, and a 2716 is used as the character generator. To handle the high speed portion of the terminal the 8275 is surrounded by a small handful of TTL. The program memory is contained in one 2716 EPROM and the data and screen memory use four 2114-type RAMs.

All devices in this system are memory mapped. A bipolar PROM is used to decode all of the addresses for the RAM, ROM, 8275, and 8253. As mentioned earlier, the bipolar prom also turns $\overline{\text{READ}}$s into $\overline{\text{DACK}}$'s and $\overline{\text{WR}}$'s for the 8275. The 8255 and 8253 are decoded by a simple address line chip select method. The total package count for the system is 20, not including the serial line drivers. If this same terminal were designed using the MCS-85 family of integrated circuits, additional part savings could have been realized. The four 2114's could have been replaced by two 8185's and the 8255 and the 2716 program PROM could have been replaced by one 8755. Additionally, since both the 8185 and the 2716 have address latches no 8212 would be needed, so the total parts count could be reduced by three or four packages.

5.4 SYSTEM OPERATION

The 8085 CPU initializes each peripheral to the appropiate mode of operation following system reset. After initialization, the 8085 continually polls the 8251A to see if a character has been sent to the terminal. When a character has been received, the 8085 decodes the character and takes appropriate action. While the 8085 is executing the above "foreground" programs, it is being interrupted once every 617 microseconds by the 8275. This "background" program is used to load the row buffers on the 8275. The 8085 is also interrupted once every frame time, or 16.67 ms, to read the keyboard and the status of the 8275.

As discussed earlier, a special POP technique was used to rapidly move the contents of the display RAM into the 8275's row buffers. The characters are then synchronously transferred to the character code outputs CC0-CC6, connected to the character generator address lines A3-A9 (Figure 5.3). Line count outputs LC0-LC2 from the 8275 are applied to the character generator address lines A0-A2. The 8275 displays character rows one line at a time. The line count outputs are used to determine which line of the character selected by A3-A8 will be displayed. Following the transfer of the first line to the dot timing logic, the line count is incremented and the second line of the character row is selected. This process continues until the last line of the row is transferred to the dot timing logic.

The dot timing logic latches the output of the character generator ROM into a parallel in,serial out synchronous shift register. This shift register is clocked at the dot clock rate (11.34 MHz) and its output constitutes the video input to the CRT.

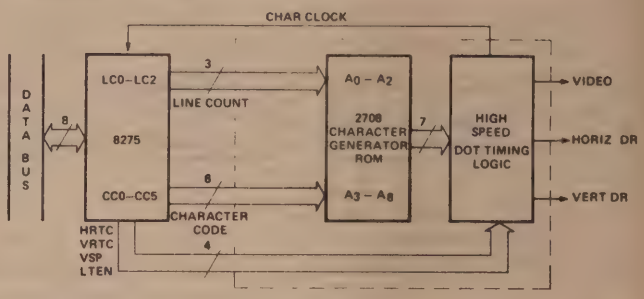


Figure 5-3 Character Generator/Dot Timing Logic Block Diagram

Table 5-1

PARAMETER	RANGE
Vertical Blanking Time (VRTC)	900 μsec nominal
Vertical Drive Pulsewidth	300 μsec ⩽ PW ⩽ 1.4 ms
Horizontal Blanking Time (HRTC)	11 μsec nominal
Horizontal Drive Pulsewidth	25 μsec ⩽ PW ⩽ 30 μsec
Horizontal Repetition Rate	15,750 ±500 pps

5.5 SYSTEM TIMING

Before any specific timing can be calculated it is necessary to determine what constraints the chosen CRT places on the overall timing. The requirements for the Ball Bros. TV-12 monitor are shown in Table 5.1. The data from Table 5.1, the 8275 specifications, and the system target specifications are all that is needed to calculate the system's timing.

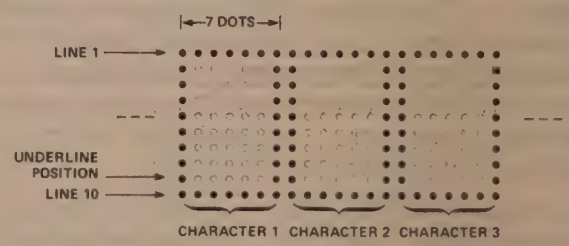


Figure 5-4. Row Format

First, let's select and "match" a few numbers. From our target specifications, we see that each character is displayed on a 7 X 10 field, and is formed by a 5 X 7 dot matrix (Figure 5.4). The 8275 allows the vertical retrace time to be only an integer multiple of the horizontal character line. This means that the total number of horizontal lines in a frame equals 10 times the number of character lines plus the vertical retrace time, which is programmed to be either 1, 2, 3, or 4 character lines. Twenty-five display lines

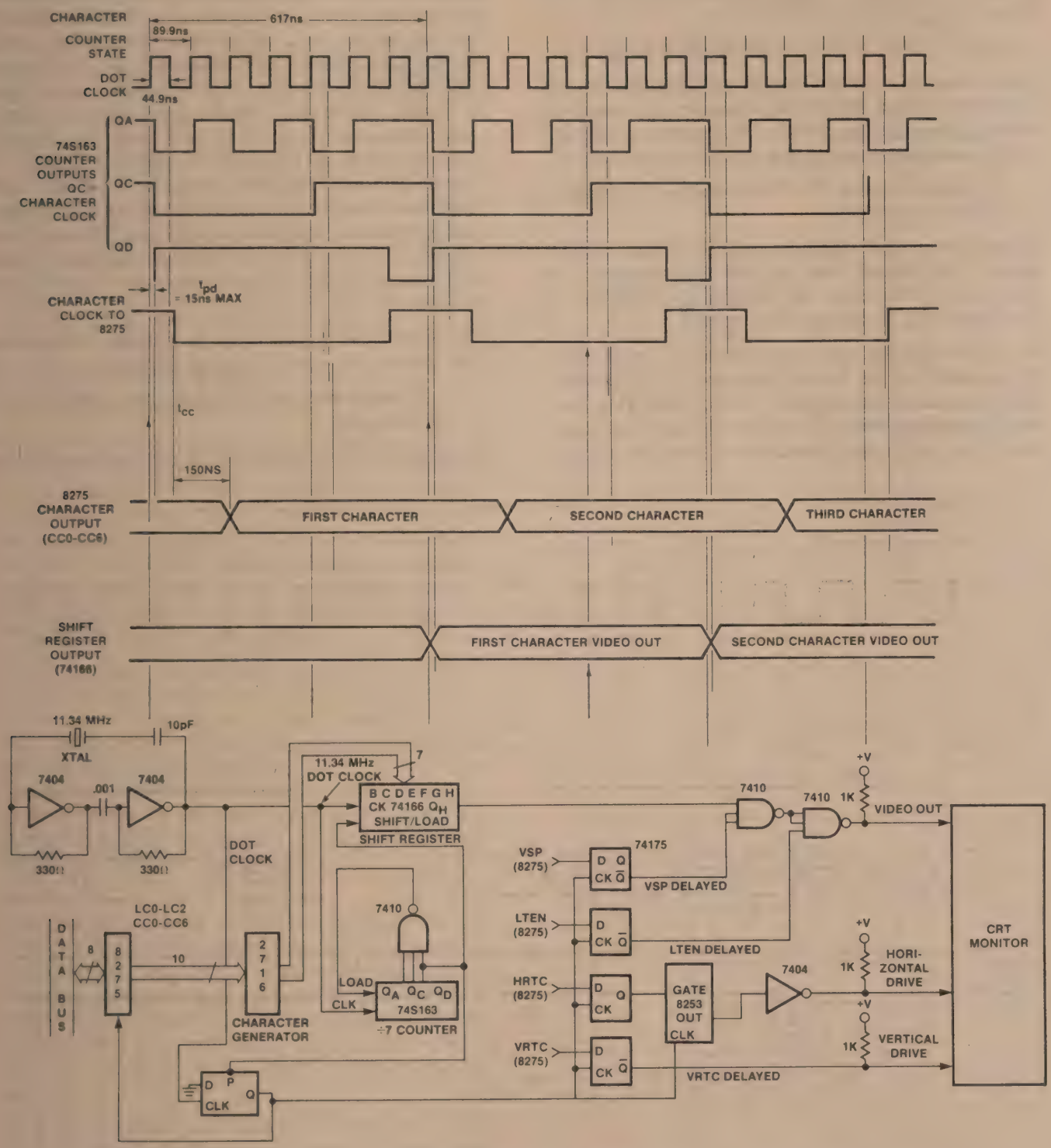


Figure 5-5. Dot Timing Logic

require 250 horizontal lines. So, if we wish to have a horizontal frequency in the neighborhood of 15,750 Hz we must choose either one or two character lines for vertical retrace. To allow for a little more margin at the top and bottom of the screen, two character lines were chosen for vertical retrace. This choice yields a net 250 + 20 = 270 horizontal lines per frame. So, assuming a 60 Hz frame:

60 Hz * 270 = 16,200 Hz (horizontal frequency)

This value falls within our target specification of 15,750 Hz with a 500 Hz variation and also assures timing compatibility with the Ball monitor since, 20 horizontal sync times yield a vertical retract time of:

61.7 microseconds X 20 horizontal sync times = 1.2345 milliseconds

This number meets the nominal VRTC and vertical drive pulse width time for the Ball monitor. A horizontal frequency of 16,200 Hz implies a 1/16,200 = 61.73 microsecond period.

It is now known that the terminal is using 250 horizontal lines to display data and 20 horizontal lines to allow for vertical retrace and that the horizontal frequency is 16,200 Hz. The next thing that needs to be determined is how much time must be allowed for horizontal retrace. Unfortunately, this number depends almost entirely on the monitor used. Usually, this number lies somewhere between 15 and 30 percent of the total horizontal line time, which in this case is 1/16,200 Hz or 61.73 microseconds. Since in most designs a fixed number of characters can be displayed on a horizontal line, it is often useful to express retrace as a given number of character times. In this design, 80 characters can be displayed on a horizontal line and it was empirically found that allowing 20 horizontal character times for retrace gave the best results. So, in reality, there are 100 character times in every given horizontal line, 80 are used to display characters and 20 are used to allow for retrace. It should be noted that if too many character times are used for retrace, less time will be left to display the characters and the display will not "fill out" the screen. Conversely, if not enough character times are allowed for retrace, the display may "run off" the screen.

One hundred character times per complete horizontal line means that each character requires

61.73 microseconds / 100 character times = 617.3 nanoseconds.

If we multiply the 20 horizontal retrace times by the

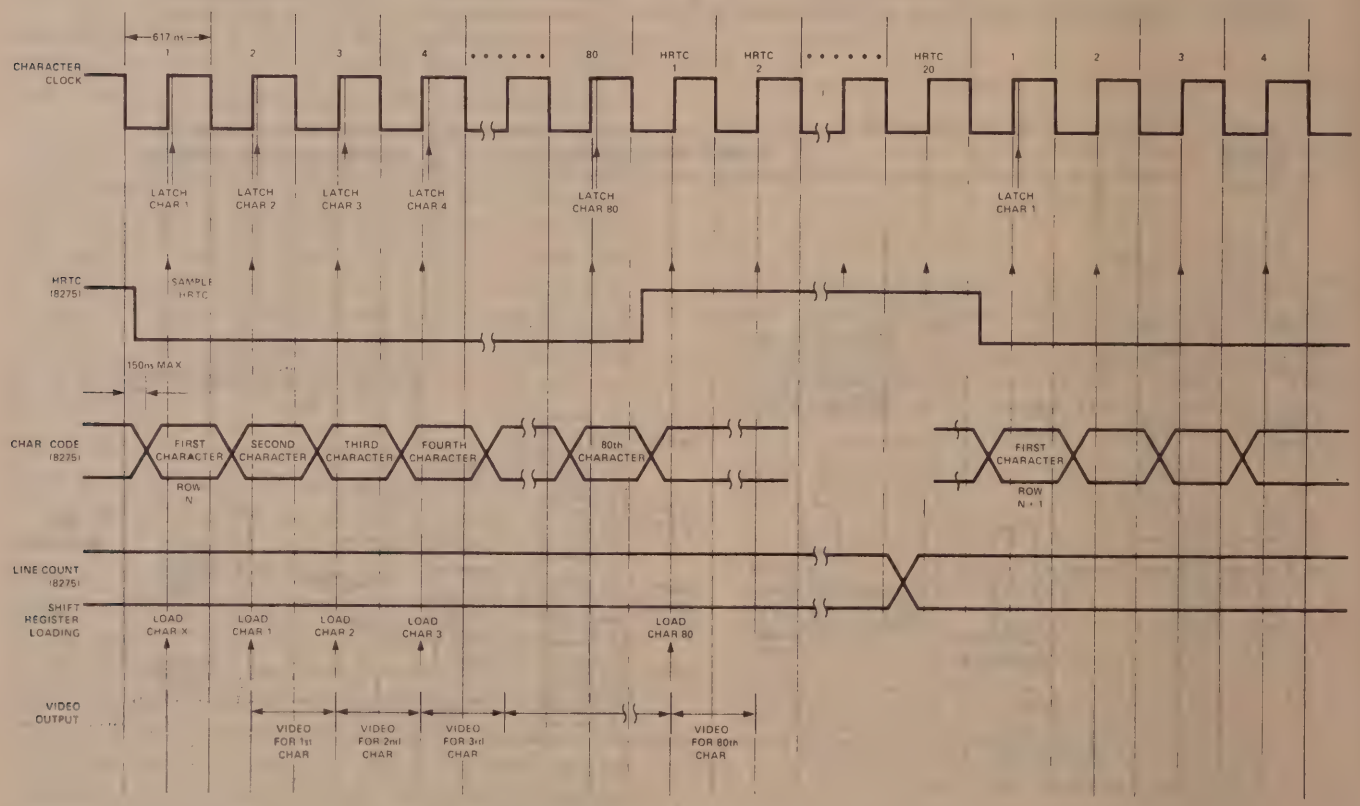


Figure 5-6. CRT System Timing

617.3 nanoseconds needed for each character, we find

617.3 nanoseconds * 20 retrace times = 12.345 microseconds

This value falls short of the 25 to 30 microseconds required by the horizontal drive of the Ball monitor. To correct for this, an 8253 was programmed in the one-shot mode and was used to extend the horizontal drive pulsewidth.

Now that the 617.3 nanosecond character clock period is known, the dot clock is easy to calculate. Since each character is formed by placing 7 dots along the horizontal.

DOT CLOCK PERIOD = 617.3 ns (CHARACTER CLK PERIOD)/ 7 DOTS

DOT CLOCK PERIOD = 88.183 nanoseconds

DOT CLOCK FREQUENCY = 1/PERIOD = 11.34 MHz

Figures 5.5 and 5.6 illustrate the basic dot timing and the CRT system timing, respectively.

6. SYSTEM SOFTWARE

6.1 SOFTWARE OVERVIEW

As mentioned earlier the software is structured on a "foreground-background" basis. Two interrupt-driven routines, FRAME and POPDAT (Fig. 6.1) request service every 16.67 milliseconds and 617 microseconds respectively, frame is used to check the baud rate switches, update the system pointers and decode and assemble the keyboard characters. POPDAT is used to move data from the memory into the 8275's row buffer rapidly.

The foreground routine first examines the line-local switch to see whether to accept data from the USART or the keyboard. If the terminal is in the local mode, action will be taken on any data that is entered through the keyboard and the USART will be ignored on both output and input. If the terminal is in the line mode data entered through the keyboard will be transmitted by the USART and action will be taken on any data read out of the USART.

When data has been entered in the terminal the software first determines if the character received was an escape, line feed, form feed, carriage return, back space, or simply a printable character. If an escape was received the terminal assumes the next received character will be a recognizable escape sequence character. If it isn't no operation is performed.

After the character is decoded, the processor jumps to the routine to perform the required task. Figure 6.2 is a flow chart of the basic software operations; the program is listed in Appendix 6.8.

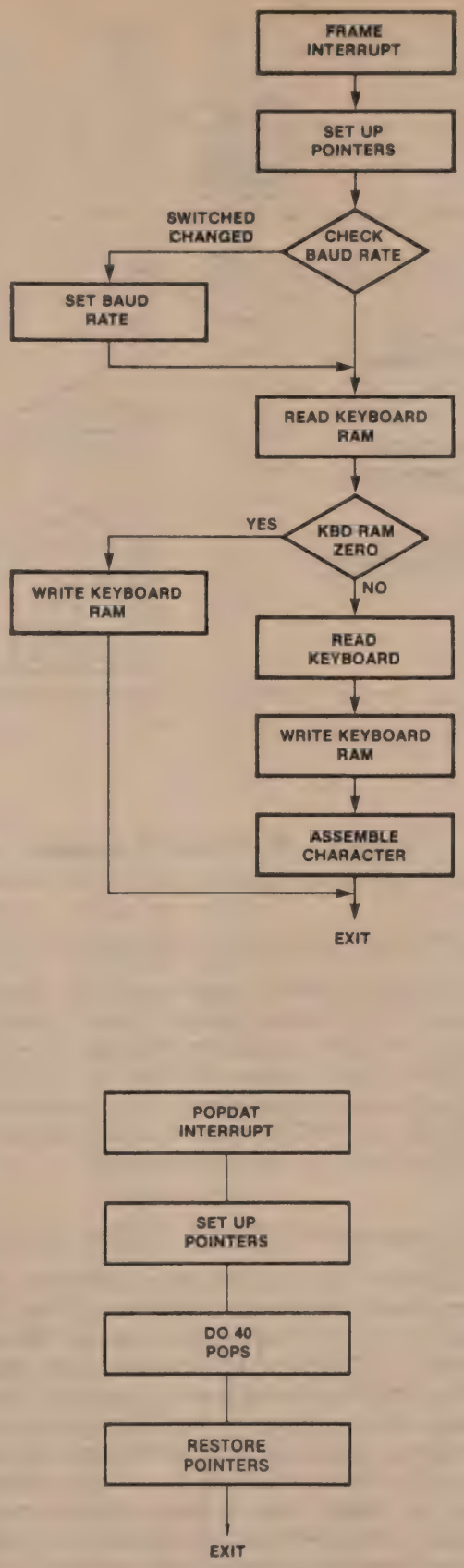


Figure 6-1. Frame and Popdat Interrupt Routines

baud. Each POPDAT interrupt executes in 211 microseconds maximum. This means that each routine must execute in:

1042 - 2 * 211 = 620 microseconds

By adding up the times for any loop, it is clear that all routines meet this speed requirement, with the exception of ESC J. This means that if the terminal is operating at 9600 baud, at least one character time must be inserted after an ESC J sequence.

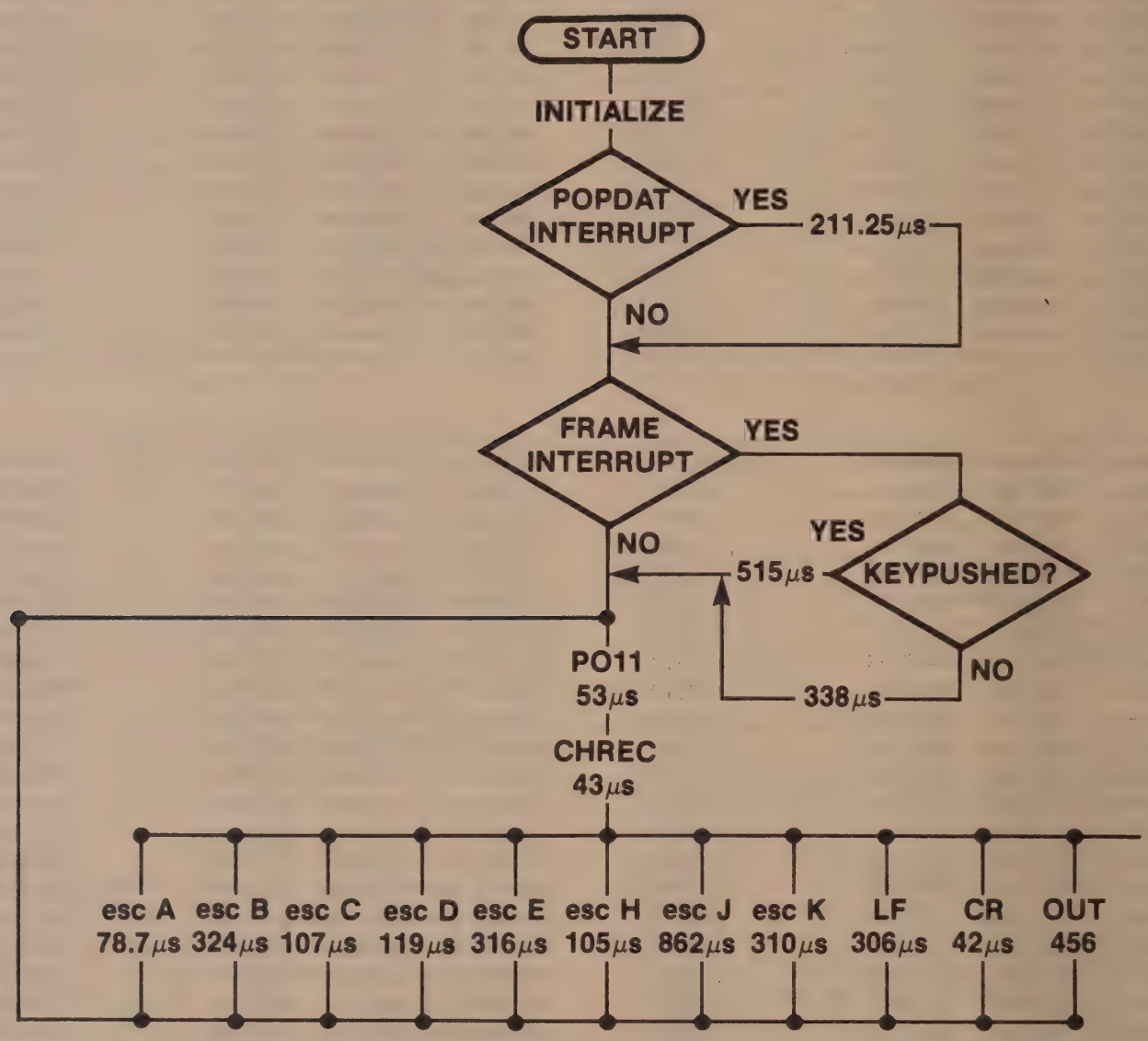
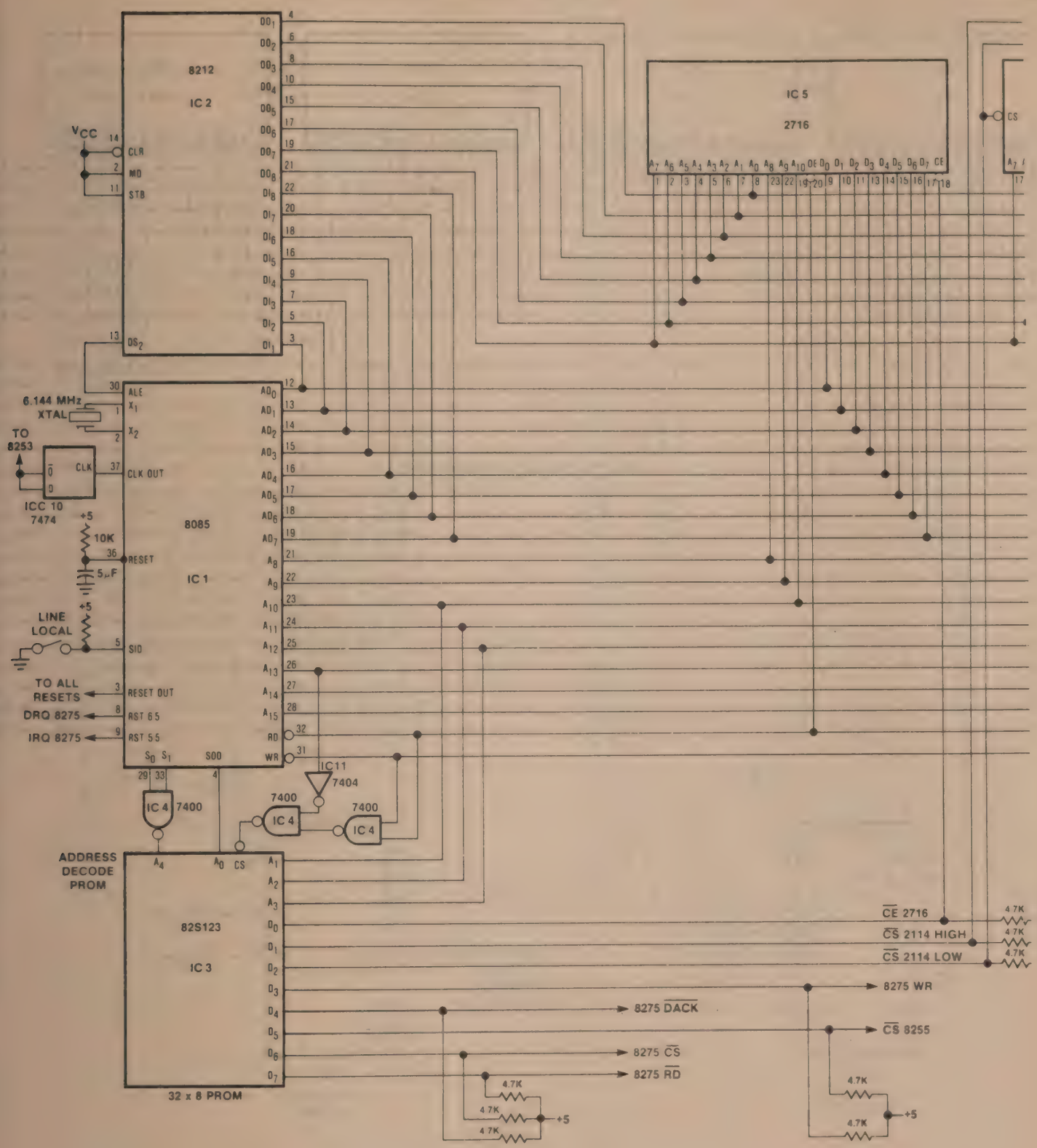
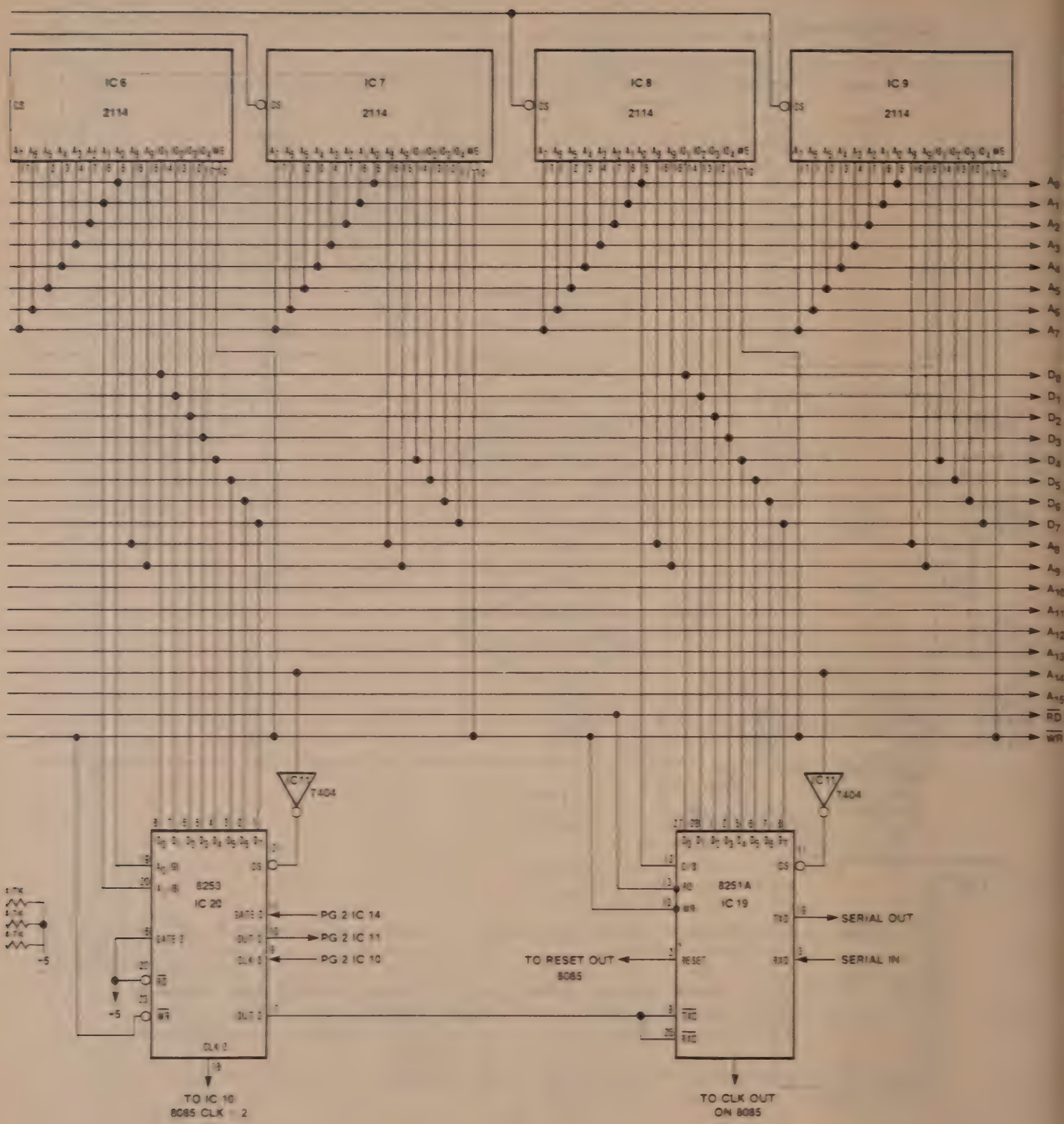
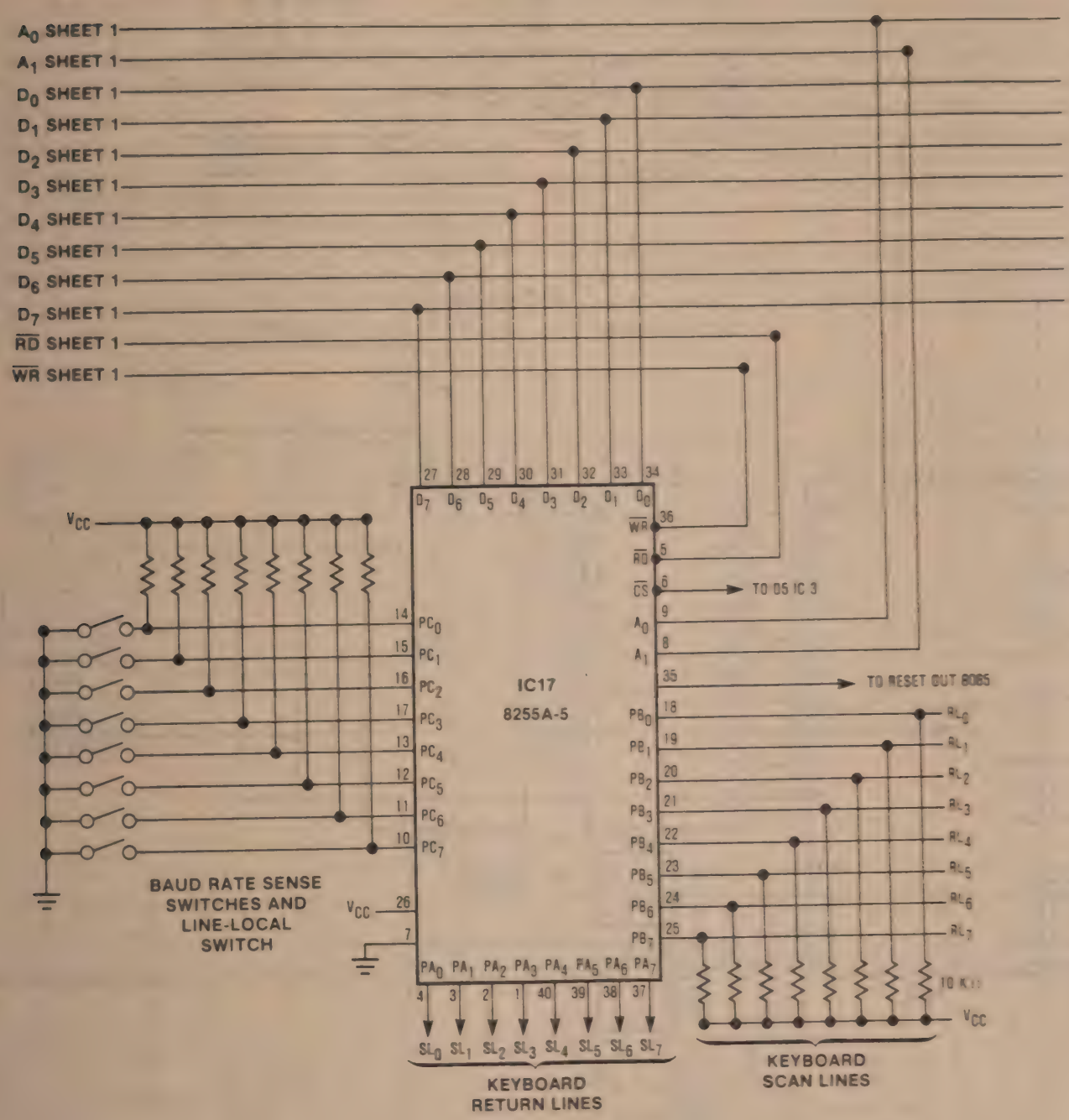


Figure 6-5. Timing Flowchart

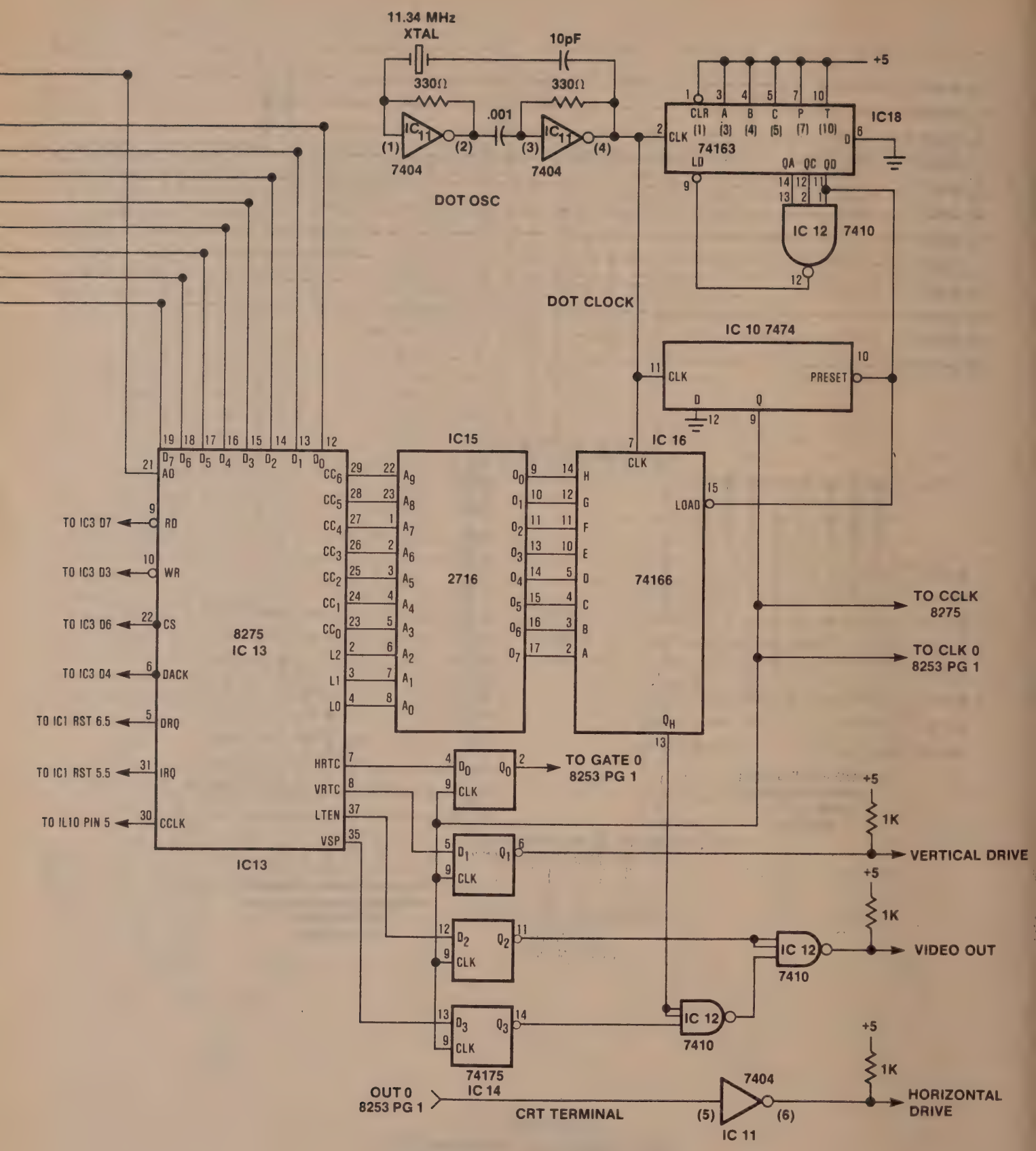


Appendix 7.1
CRT TERMINAL SCHEMATICS


IC 6
2114
IC 7
2114
IC 8
2114
IC 9
2114
A0
A1
A2
A3
A4
A5
A6
A7
D0
D1
D2
D3
D4
D5
D6
D7
A8
A9
A10
A11
A12
A13
A14
A15
RD
WR
IC 7404
IC 7404
8253
IC 20
PG 2 IC 14
PG 2 IC 11
PG 2 IC 10
-5
-5
TO IC 10
8085 CLK 2
8251A
IC 19
SERIAL OUT
SERIAL IN
TO RESET OUT
8085
TO CLK OUT
ON 8085



Appendix 7.1
CRT TERMINAL SCHEMATICS


11.34 MHz
XTAL
10pF
330Ω
330Ω
.001
IC11
7404
DOT OSC
+5
CLR A B C P T
CLK
74163
LD
QA QC QD
IC18
IC 12
7410
DOT CLOCK
IC 10 7474
CLK
PRESET
D
Q
IC15
IC 16
CLK
LOAD
2716
74166
8275
IC 13
TO IC3 D7
TO IC3 D3
TO IC3 D6
TO IC3 D4
TO IC1 RST 6.5
TO IC1 RST 5.5
TO IL10 PIN 5
RD
WR
CS
DACK
DRQ
IRQ
CCLK
AD
HRTC
VRTC
LTEN
VSP
IC13
TO CCLK
8275
TO CLK 0
8253 PG 1
TO GATE 0
8253 PG 1
1K
VERTICAL DRIVE
VIDEO OUT
74175
IC 14
OUT 0
8253 PG 1
CRT TERMINAL
IC 11
HORIZONTAL
DRIVE

207780-001

Appendix 7.2
KEYBOARD INTERFACE

The keyboard used in this design was a simple unencoded ASCII keyboard. In order to keep the cost to a minimum a simple scan matrix technique was implemented by using two ports of an 8255 parallel I/O device.

When the system is initialized the contents of the eight keyboard RAM locations are set to zero. Once every frame, which is 16.67 milliseconds the contents of the keyboard ram is read and then rewritten with the contents of the current switch matrix. If a non-zero value of one of the keyboard RAM locations is found to be the same as the corresponding current switch matrix, a valid key push is registered and action is taken. By operating the keyboard scan in this manner an automatic debounce time of 16.67 milliseconds is provided.

Figure 7.2A shows the actual physical layout of the keyboard and Figure 7.2B shows how the individual keys were encoded. On Figure 7.2B the scan lines are the numbers on the bottom of each key position and the return lines are the numbers at the top of each key position. The shift, control, and caps lock key were brought in through separate lines of port C of the 8255. Figure 7.3 shows the basic keyboard matrix.

In order to guarantee that two scan lines could not be shorted together if two or more keys are pushed simultaneously, isolation diodes could be added as shown in Figure 7.4.

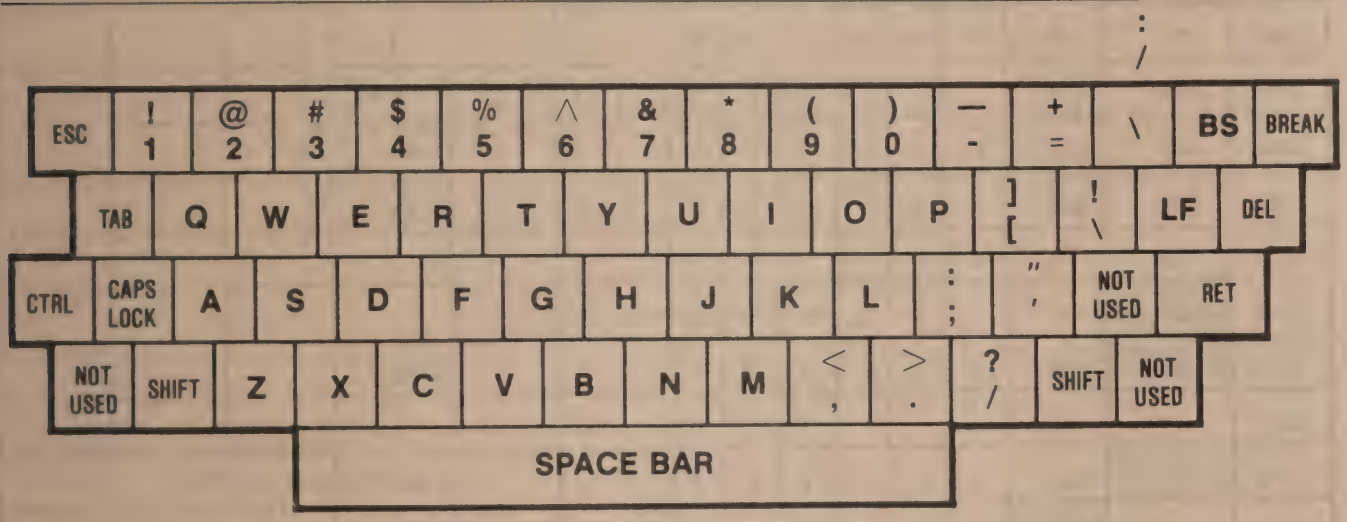
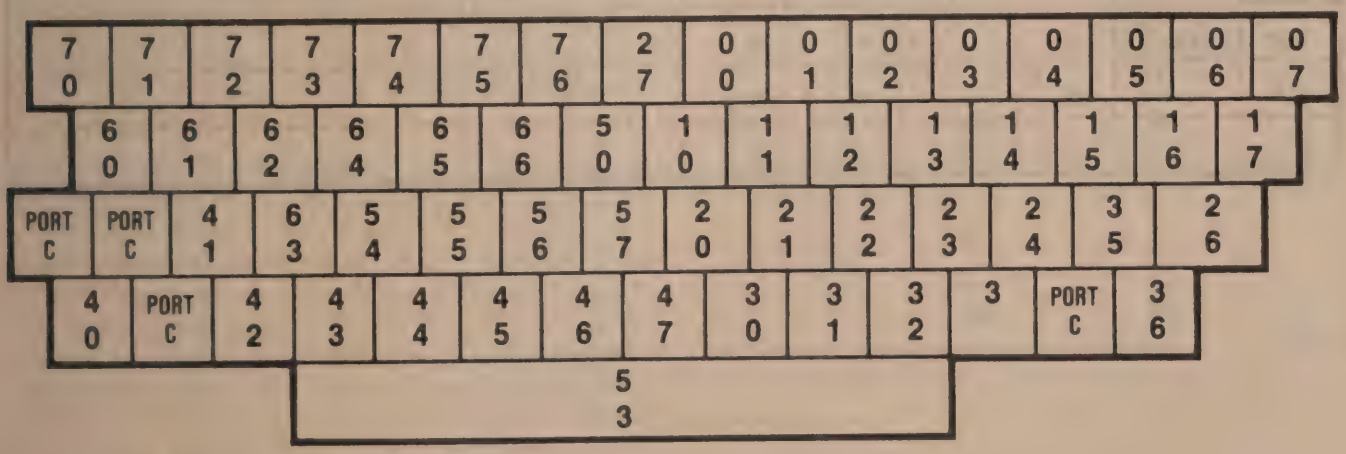


Figure 7-2A. Keyboard Layout



TOP NUMBER = RETURN LINE

BOTTOM NUMBER = SCAN LINE

Figure 7-2B. Keyboard Encoding

Appendix 7.3
ESCAPE/CONTROL/DISPLAY CHARACTER SUMMARY

	CONTROL CHARACTERS		DISPLAYABLE CHARACTER								ESCAPE SEQUENCE			
BIT	0_0	0_1	0_0	0_1	1_0	1_1	1_0	1_1	0_0	0_1	1_0	1_1	1_0	1_1
0000	NUL @	DLE P	SP	ϕ	@	P		P						
0001	SOH A	DCI Q	!	I	A	Q	A	Q			↑ A			
0010	STX B	DC2 R	"	2	B	R	B	R			↓ B			
0011	ETX C	DC3 S	#	3	C	S	C	S			→ C			
0100	EOT D	DC4 T	$	4	D	T	D	T			← D			
0101	ENQ E	NAK U	%	5	E	U	E	U			CLR E			
0110	ACK F	SYN V	&	6	F	V	F	V						
0111	BEL G	ETB W	'	7	G	W	G	W						
1000	BS H	CAN X	(	8	H	X	H	X			HOME H			
1001	HT I	EM Y	)	9	I	Y	I	Y						
1010	LF J	SUB Z	*	:	J	Z	J	Z			EOS I			
1011	VT K	ESC [	+	;	K	[	K				EL J			
1100	FF L	FS /	,	<	L	\	L							
1101	CR M	GS	–	=	M	]	M							
1110	SO N	RS ^	.	>	N	^	N							
1111	S1 O	US –	/	?	O	–	O							

NOTE: Shaded blocks - functions terminal will react to. Others can be generated but are ignored up on receipt.

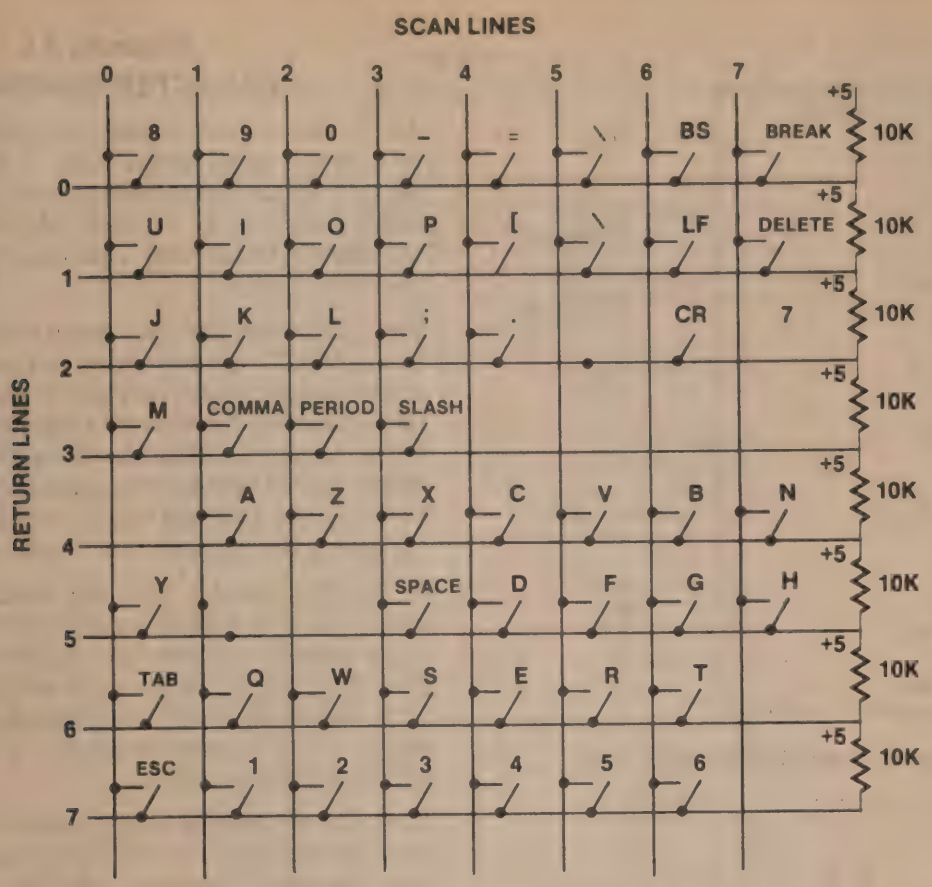


Figure 7-3. Keyboard Matrix

Appendix 7.4
PROM DECODING

As stated earlier, all of the logic necessary to convert the 8275 into a non-DMA type of device was performed by a single small bipolar prom. Besides turning certain processor READS into DACKS and WRITES for the 8275, this 32 by 8 prom decoded addresses for the system ram, rom, as well as for the 8255 parallel I/O port.

Any bipolar prom that has a by eight configuration could function in this application. This particular device was chosen simply because it is the only "by eight" prom available in a 16 pin package. The connection of the prom is shown in detail in Figure 7.5 and its truth table is shown in Figure 7.6. Note that when a fetch cycle (M1) is not being performed, the state of the SOD line is the only thing that determines if memory reads will be written into the 8275's row buffers. This is done by pulling both DACK and WRITE low on the 8275.

Also note that all of the outputs of the bipolar prom MUST BE PULLED HIGH by a resistor. This prevents any unwanted assertions when the prom is disabled.

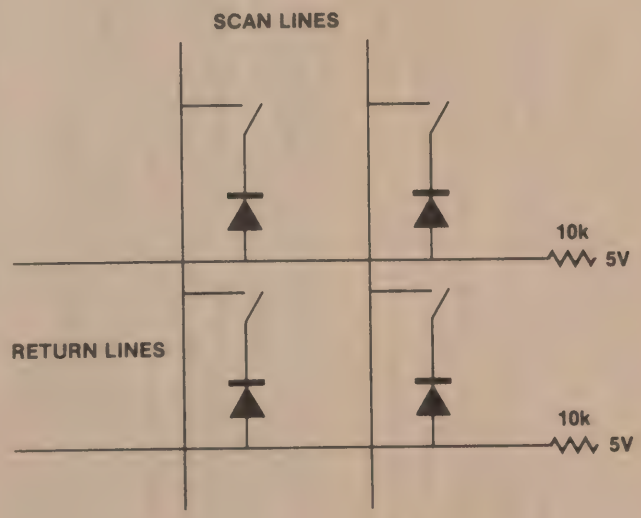


Figure 7-4. Isolating Scan Lines With Diodes

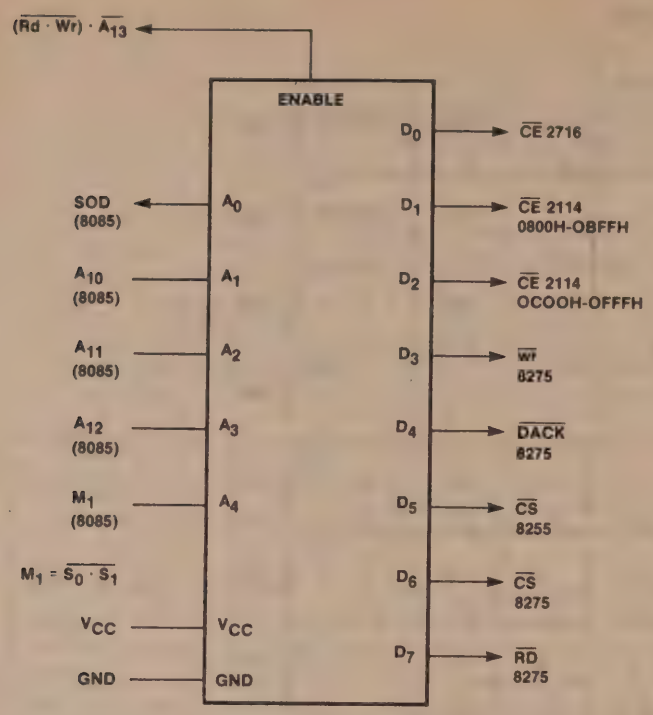


Figure 7-5. Bipolar Prom (825123) Connection

M1	A12	A11	A10	Sod	8275 $\overline{Rd}$	8275 $\overline{CS}$	8255 $\overline{CS}$	8275 $\overline{DACK}$	8275 WR	2114 H	2114 L	2716
A4	A3	A2	A1	A0	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	1	1	1	1	1	1	1	0
0	0	0	0	1	1	1	1	1	1	1	1	0
0	0	0	1	0	1	1	1	1	1	1	1	0
0	0	0	1	1	1	1	1	1	1	1	1	0
0	0	1	0	0	1	1	1	1	1	1	0	1
0	0	1	0	1	1	1	1	1	1	1	0	1
0	0	1	1	0	1	1	1	1	1	0	1	1
0	0	1	1	1	1	1	1	1	1	0	1	1
0	1	0	0	0	1	0	1	1	0	1	1	1
0	1	0	0	1	1	0	1	1	0	1	1	1
0	1	0	1	0	0	0	1	1	1	1	1	1
0	1	0	1	1	0	0	1	1	1	1	1	1
0	1	1	0	0	1	1	0	1	1	1	1	1
0	1	1	0	1	1	1	0	1	1	1	1	1
0	1	1	1	0	1	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	1	1	1	1	1
1	0	0	0	0	1	1	1	1	1	1	1	0
1	0	0	0	1	1	1	1	0	0	1	1	0
1	0	0	1	0	1	1	1	1	1	1	1	0
1	0	0	1	1	1	1	1	0	0	1	1	0
1	0	1	0	0	1	1	1	1	1	1	0	1
1	0	1	0	1	1	1	1	0	0	1	0	1
1	0	1	1	0	1	1	1	1	1	0	1	1
1	0	1	1	1	1	1	1	0	0	0	1	1
1	1	0	0	0	1	0	1	1	0	1	1	1
1	1	0	0	1	1	0	1	1	0	1	1	1
1	1	0	1	0	0	0	1	1	1	1	1	1
1	1	0	1	1	0	0	1	1	1	1	1	1
1	1	1	0	0	1	1	0	1	1	1	1	1
1	1	1	0	1	1	1	0	1	1	1	1	1
1	1	1	1	0	1	1	0	1	1	1	1	1
1	1	1	1	1	1	1	0	1	1	1	1	1

Figure 7-6. Truth Table Bipolar Prom

Appendix 7.5
CHARACTER GENERATOR

As previously mentioned, the character generator used in this terminal is a 2716 or 2758 EPROM. A 1K by 8 device is sufficient since a 128 character 5 by 7 dot matrix only requires 8K of memory. Any "standard" or custom character generator could have been used.

The three low-order line count outputs (LC0-LC2) from the 8275 are connected to the three low-order address lines of the character generator and the seven character generator outputs (CC0-CC6) are connected to A3-A9 of the character generator. The output from the character generator is loaded into a shift register and the serial output from the shift register is the video output of the terminal.

Now, let's assume that the letter "E" is to be displayed. The ASCII code for "E" is 45H. So, 45H is presented to address lines A2-A9 of the character generator. The scan lines will now count each line from zero to seven to "form" the character as shown in Fig. 7.7. This same procedure is used to form all 128 possible characters.

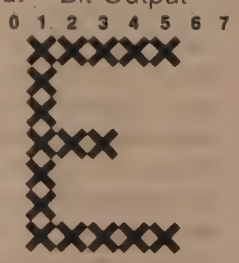
It should be obvious that "custom" character fonts could be made just by changing the bit patterns in the character generator PROM. For reference, Appendix 7.6 contains a HEX dump of the character generator used in this terminal.

45H = 01000101
Address to Prom = 01000101 SL2 SL1 SL0
= 228H - 22FH

Depending on state of Scan lines.

Character generator output

Rom Address	Rom Hex Output	Bit Output*
228H	3E	
229H	02	
22AH	02	
22BH	0E	
22CH	02	
22DH	02	
22EH	3E	
22FH	00	



Bits 0, 6 and 7 are not used.

* note bit output is backward from convention.

Figure 7-7. Character Generation

Appendix 7.6
HEX DUMP OF CHARACTER GENERATOR

```
:1000000001000000000000000000000000000000EF
:1000100000000000000000000000000000000000E0
:1000200000000000000000000000000000000000D0
:1000300000000000000000000000000000000000C0
:1000400000000000000000000000000000000000B0
:1000500000000000000000000000000000000000A0
:1000600000000000000000000000000000000000A0
:1000700000000000000000000000000000000000A0
:1000800000000000000000000000000000000000A0
:1000900000000000000000000000000000000000A0
:1000A00000000000000000000000000000000000A0
:1000B00000000000000000000000000000000000A0
:1000C00000000000000000000000000000000000A0
:1000D00000000000000000000000000000000000A0
:1000E00000000000000000000000000000000000A0
:1000F00000000000000000000000000000000000A0
:1001000000000000000000000000000000000000A0
:1001100014141400000000000014143E143E141400C3
:1001200000083C0A1C281E08000062610408043230005D
:1001300000400A0A042A122C000080800000000000023
:1001400000080402020204080000810202020100080001
:1001500000082A1C081C2A08000018083E08080000009D
:1001600000000000000000080804000000003C000000003F
:1001700000000000000000180000201008040200000029
:1001800001C22322A26221C000080C0008080181C00021
:1001900001C22201C0202 3E003E201018202221C008F
:1001A00001018142 3E101000 3E021E202022 1C00C7
:1001B00003804021E22221C003E201008040400000001
:1001C00001C22221C22221C001C22223C201 0E00379
:1001D00000000000000000000000000000000000000000
:1001E00001008040204081000000003E003E000000059
:1001F00000040810201008040001C222010008080000921
:1002000001C222A3A1A023C000081422223E22220012
:1002100001E24241C24241E001C22202022221C00374
:1002200001E24242424241E003E02020E02023E004C
:1002300003E42020E02020200 3C02023A22223C0065E
:1002400002222223E222222001C008080808181C0044
:1002500007020202020221C002212020A050A122200EE
:1002600002020202020223E002223 52A2A2222220032
:1002700022262A322222224001C222222222221C00392
:1002800001E22221E020202001C2222222A122C00FE
:1002900001E22221E0A122200 3C02021C20201E00E5
:1002A00003E0808080808080222222222222221C00F8
:1002B00022222222221408002222222222A3522005E
:1002C00022221408142222002222221408080808 0E4
:1002D00003E2010080402 3E001C040404040041C0018
:1002E00000020408102000003820202020330 00C0
:1002F00001C2A080808080000000000000007E12
:1003000018901100000000000000 3C243C223C004E
:1003100002021A2622221E000000038040400438003B
:1003200020202C322222 3C000000038248C84B8 0005B
:1003300038240400E040404000000 3C22223C203CAB
:1003400002021A2622222200080000808080 8390004B
:1003500020002020A42418020222120A162200C3
:1003600088080808080 8390000000 3622A2A2222007F
:1003700000001A2622222200000018242424180003B
:1003800000001E22221E020200001C22223C202000D
:1003900000001A2620202020000003804 1820 1C0087
:1003A00003081C0808089000000002222222324C0095
:1003B00000022222214080000002222 2A3E1400E3
:1003C00000022140814220000002222222 3C2438BF
:1003D00003E100804 3E001888890388891900 42F
:1003E00080808080808080808 3C9091219090 0D0051
:1003F00000008C2380018000000000000000000095
```

Appendix 7.7
COMPOSITE VIDEO

In this design, it was assumed that the monitor required a separate horizontal drive, vertical drive, and video input. However, many monitors require a composite video signal. The schematic shown in Figure 7.8 illustrates how to generate a composite video signal from the output of the 8275.

The dual one-shots are used to provide a small delay and the proper horizontal and vertical pulse to the composite video monitor. The delay introduced in the vertical and horizontal timing is used to "center" the display. VR1 and VR2 control the amount of delay. IC3 is used to mix the vertical and horizontal retrace and Q1 along with the R1, R2, and R3 mix the video and the retrace signal and provide the proper DC levels.

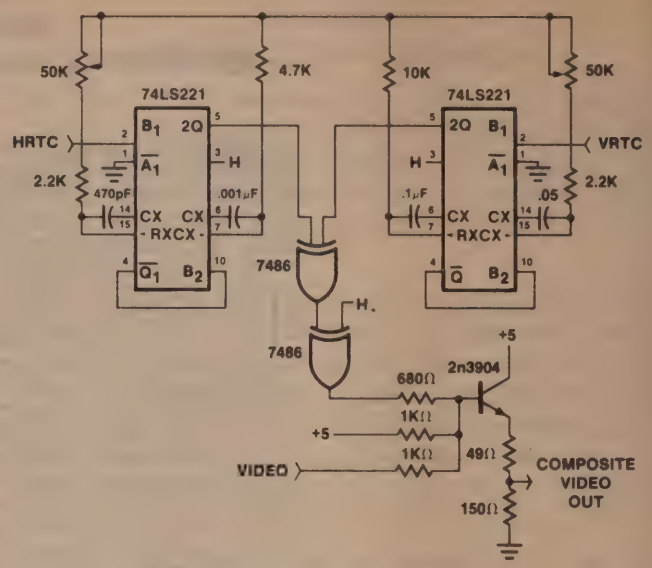


Figure 7-8. Composite Video

Appendix 7.8
SOFTWARE LISTINGS

```
ISIS-II 8080/8085 MACRO ASSEMBLER, X108

LOC  OBJ          SEQ        SOURCE STATEMENT

                    1 $MOD85  MACROFILE
                    2         ;NO DMA 8275 SOFTWARE ALL I/O IS MEMORY MAPPED
                    3         ;SYSTEM ROM 0000H TO 07FFH
                    4         ;SYSTEM RAM 0800H TO 0FFFH
                    5         ;8275 WRITE 1000H TO 13FFH
                    6         ;8275 READ 1400H TO 17FFH
                    7         ;8255 READ/WRITE 1800H TO 1FFF
                    8         ;8253 ENABLED BY A14
                    9         ;8251 ENABLED BY A15
1800               10 PORTA   EQU     1800H          ;8255 PORT A ADDRESS
1801               11 PORTB   EQU     1801H          ;8255 PORT B ADDRESS
1802               12 PORTC   EQU     1802H          ;8255 PORT C ADDRESS
1803               13 CNWD55  EQU     1803H          ;8255 CONTROL PORT ADDRESS
A001               14 USTF    EQU     0A001H         ;8251 FLAGS
A000               15 USTD    EQU     0A000H         ;8251 DATA
6000               16 CNT0    EQU     6000H          ;8253 COUNTER 0
6001               17 CNT1    EQU     6001H          ;8253 COUNTER 1
6002               18 CNT2    EQU     6002H          ;8253 COUNTER 2
6003               19 CNTM    EQU     6003H          ;8253 MODE WORD
1001               20 CRTS    EQU     1001H          ;8275 CONTROL ADDRESS
1000               21 CRTM    EQU     1000H          ;8275 MODE ADDRESS
1401               22 INT75   EQU     1401H          ;8275 INTERRUPT CLEAR
0800               23 TPDIS   EQU     0800H          ;TOP OF DISPLAY RAM
0F80               24 BTDIS   EQU     0F80H          ;BOTTOM OF DISPLAY RAM
0FD0               25 LAST    EQU     0FD0H          ;FIRST BYTE AFTER DISPLAY
0018               26 CURBOT  EQU     18H            ;BOTTOM Y CURSOR
0050               27 LNGTH   EQU     0050H          ;LENGTH OF ONE LINE
0FE0               28 STPTR   EQU     0FE0H          ;LOCATION OF STACK POINTER
                   29         ;
                   30         ;START PROGRAM
                   31         ;ALL VARIABLES ARE INITIALIZED BEFORE ANYTHING ELSE
                   32         ;
0000 F3            33         DI                     ;DISABLE INTERRUPTS
0001 31E00F        34         LXI     SP,STPTR       ;LOAD STACK POINTER
0004 210008        35         LXI     H,TPDIS        ;LOAD H&L WITH TOP OF DISPLAY
0007 22E30F        36         SHLD    TOPAD          ;SET TOP = TOP OF DISPLAY
000A 22E80F        37         SHLD    CURAD          ;STORE THE CURRENT ADDRESS
000D 3E00          38         MVI     A,00H          ;ZERO A
000F 32E10F        39         STA     CURSY          ;ZERO CURSOR Y POINTER
0012 32E20F        40         STA     CURSX          ;ZERO CURSOR X POINTER
0015 32EB0F        41         STA     KBCHR          ;ZERO KBD CHARACTER
0018 32E70F        42         STA     USCHR          ;ZERO USART CHAR BUFFER
001B 32EA0F        43         STA     KEYDWN         ;ZERO KEY DOWN
```

```
001E 32ED0F          STA     KEYOK           ;ZERO KEYOK
0021 32EE0F          STA     ESCP            ;ZERO ESCAPE
0024 C39800          JMP     LPKBD           ;JUMP AND SET EVERYTHING UP
                     ;
                     ;THIS JUMP VECTOR IS LOCATED AT THE RST 5.5 LOCATION
                     ;OF THE 8085. IT IS USED TO READ THE 8275 STATUS AND
                     ;READ THE KEYBOARD. THIS ROUTINE IS EXECUTED ONCE EVERY
                     ;16.667 MILLISECONDS.
                     ;
002C                 ORG     002CH
002C C36701          JMP     FRAME
                     ;
                     ;THIS ROUTINE IS LOCATED AT THE RST 6.5 LOCATION OF THE
                     ;8085 AND IS USED TO LOAD THE DATA TO BE DISPLAYED INTO
                     ;THE 8275. THIS ROUTINE IS EXECUTED ONCE EVERY 617 MICROSECONDS.
                     ;
0034                 ORG     34H
0034 F5      POPDAT: PUSH    PSW             ;SAVE A AND FLAGS
0035 E5              PUSH    H               ;SAVE H AND L
0036 D5              PUSH    D               ;SAVE D AND E
0037 210000          LXI     H,0000H         ;ZERO H AND L
003A 39              DAD     SP              ;PUT STACK POINTER IN H AND L
003B EB              XCHG                    ;PUT STACK IN D AND E
003C 2AE80F          LHLD    CURAD           ;GET POINTER
003F F9              SPHL                    ;PUT CURRENT LINE INTO SP
0040 3EC0            MVI     A,0C0H          ;SET MASK FOR SIM
0042 30              SIM
                     REPT (LNGTH/2)
                     POP     H
                     ENDM
0043 E1              POP     H
0044 E1              POP     H
0045 E1              POP     H
0046 E1              POP     H
0047 E1              POP     H
0048 E1              POP     H
0049 E1              POP     H
004A E1              POP     H
004B E1              POP     H
004C E1              POP     H
004D E1              POP     H
004E E1              POP     H
004F E1              POP     H
0050 E1              POP     H
0051 E1              POP     H
0052 E1              POP     H
0053 E1              POP     H
0054 E1              POP     H
0055 E1              POP     H
0056 E1              POP     H
0057 E1              POP     H
0058 E1              POP     H
0059 E1              POP     H
005A E1              POP     H
005B E1              POP     H
005C E1              POP     H
005D E1              POP     H
005E E1              POP     H
005F E1              POP     H
0060 E1              POP     H
0061 E1              POP     H
0062 E1              POP     H
0063 E1              POP     H
0064 E1              POP     H
0065 E1              POP     H
0066 E1              POP     H
0067 E1              POP     H
0068 E1              POP     H
0069 E1              POP     H
006A E1              POP     H
006B 0F              RRC                     ;SET UP A
006C 30              SIM                     ;GO BACK TO NORMAL MODE
006D 210000          LXI     H,0000H         ;ZERO HL
0070 39              DAD     SP              ;ADD STACK
0071 EB              XCHG                    ;PUT STACK IN H AND L
0072 F9              SPHL                    ;RESTORE STACK
0073 21D00F          LXI     H,LAST          ;PUT BOTTOM DISPLAY IN H AND L
0076 EB              XCHG                    ;SWAP REGISTERS
0077 7A              MOV     A,D             ;PUT HIGH ORDER IN A
0078 BC              CMP     H               ;SEE IF SAME AS H
0079 C28400          JNZ     KPTK            ;IF NOT LEAVE
007C 7B              MOV     A,E             ;PUT LOW ORDER IN A
007D BD              CMP     L               ;SEE IF SAME AS L
007E C28400          JNZ     KPTK            ;IF NOT LEAVE
0081 210008          LXI     H,TPDIS         ;LOAD H AND L WITH TOP OF SCREEN MEMORY
0084 22E80F  KPTK:   SHLD    CURAD           ;PUT BACK CURRENT ADDRESS
0087 3E18            MVI     A,18H           ;SET MASK
0089 30              SIM                     ;OUTPUT MASK
```

```
008A D1                 POP     D               ;GET D AND E
008B E1                 POP     H               ;GET H AND L
008C F1                 POP     PSW             ;GET A AND FLAGS
008D FB                 EI                      ;TURN ON INTERRUPTS
008E C9                 RET                     ;GO BACK
                        ;
                        ;THIS IS THE EXIT ROUTINE FOR THE FRAME INTERRUPT
                        ;
008F 3E18       BYPASS: MVI     A,18H           ;SET MASK
0091 30                 SIM                     ;OUTPUT THE MASK
0092 C1                 POP     B               ;GET B AND C
0093 D1                 POP     D               ;GED D AND E
0094 E1                 POP     H               ;GET H AND L
0095 F1                 POP     PSW             ;GET A AND FLAGS
0096 FB                 EI                      ;ENABLE INTERRUPTS
0097 C9                 RET                     ;GO BACK
                        ;
                        ;THIS CLEARS THE AREA OF RAM THAT IS USED
                        ;FOR KEYBOARD DEBOUNCE.
                        ;
0098 32EF0F     LPKBD:  STA     SHCON           ;ZERO SHIFT CONTROL
009B 32F00F             STA     RETLIN          ;ZERO RETURN LINE
009E 32F10F             STA     SCNLIN          ;ZERO SCAN LINE
                        ;
                        ;THIS ROUTINE CLEARS THE ENTIRE SCREEN BY PUTTING
                        ;SPACE CODES (20H) IN EVERY LOCATION ON THE SCREEN.
                        ;
00A1 210008             LXI     H,TPDIS         ;PUT TOP OF SCREEN IN HL
00A4 01D00F             LXI     B,LAST          ;PUT BOTTOM IN BC
00A7 3620       LOOPF:  MVI     M,20H           ;PUT SPACE IN M
00A9 23                 INX     H               ;INCREMENT POINTER
00AA 7C                 MOV     A,H             ;GET H
00AB B8                 CMP     B               ;SEE IF SAME AS B
00AC C2A700             JNZ     LOOPF           ;IF NOT LOOP AGAIN
00AF 7D                 MOV     A,L             ;GET L
00B0 B9                 CMP     C               ;SEE IF SAME AS C
00B1 C2A700             JNZ     LOOPF           ;IF NOT LOOP AGAIN
                        ;
                        ;8255 INITIALIZATION
                        ;
00B4 3E8B               MVI     A,8BH           ;MOVE 8255 CONTROL WORD INTO A
00B6 320318             STA     CNWD55          ;PUT CONTROL WORD INTO 8255
                        ;
                        ;8251 INITIALIZATION
                        ;
00B9 2101A0             LXI     H,USTF          ;GET 8251 FLAG ADDRESS
00BC 3680               MVI     M,80H           ;DUMMY STORE TO 8251
00BE 3600               MVI     M,00H           ;RESET 8251
00C0 3640               MVI     M,40H           ;RESET 8251
00C2 00                 NOP                     ;WAIT
00C3 36EA               MVI     M,0EAH          ;LOAD 8251 MODE WORD
00C5 3605               MVI     M,05H           ;LOAD 8251 COMMAND WORD
                        ;
                        ;8253 INITIALIZATION
                        ;
00C7 3E32               MVI     A,32H           ;CONTROL WORD FOR 8253
00C9 320360             STA     CNTM            ;PUT CONTROL WORD INTO 8253
00CC 3E32               MVI     A,32H           ;LSB 8253
00CE 320060             STA     CNT0            ;PUT IT IN 8235
00D1 3E00               MVI     A,00H           ;MSB 8253
00D3 320060             STA     CNT0            ;PUT IT IN 8253
00D6 CDDC00             CALL    STBAUD          ;GO DO BAUD RATE
00D9 C3F900             JMP     IN75            ;GO DO 8275
                        ;
                        ;THIS ROUTINE READS THE BAUD RATE SWITCHES FROM PORT C
                        ;OF THE 8255 AND LOOKS UP THE NUMBERS NEEDED TO LOAD
                        ;THE 8253 TO PROVIDE THE PROPER BAUD RATE.
                        ;
00DC 3A0218     STBAUD: LDA     PORTC           ;READ BAUD RATE SWITCHES
00DF E60F               ANI     0FH             ;STRIP OFF 4 MSB'S
00E1 32EC0F             STA     BAUD            ;SAVE IT
00E4 07                 RLC                     ;MOVE BITS OVER ONE PLACE
00E5 21C505             LXI     H,BDLK          ;GET BAUD RATE LOOK UP TABLE
00E8 1600               MVI     D,00H           ;ZERO D
00EA 5F                 MOV     E,A             ;PUT A IN E
00EB 19                 DAD     D               ;GET OFFSET
00EC 110360             LXI     D,CNTM          ;POINT DE TO 8253
00EF 3EB6               MVI     A,0B6H          ;GET CONTROL WORD
00F1 12                 STAX    D               ;STORE IN 8253
00F2 1B                 DCX     D               ;POINT AT #2 COUNTER
00F3 7E                 MOV     A,M             ;GET LSB BAUD RATE
00F4 12                 STAX    D               ;PUT IT IN 8253
00F5 23                 INX     H               ;POINT AT MSB BAUD RATE
00F6 7E                 MOV     A,M             ;GET MSB BAUD RATE
00F7 12                 STAX    D               ;PUT IT IN 8253
00F8 C9                 RET                     ;GO BACK
                        ;
```

```
                    ;8275 INITIALIZATION
                    ;
00F9 210110         IN75:   LXI     H,CRTS
00FC 3600                   MVI     M,00H           ;RESET AND STOP DISPLAY
00FE 2B                     DCX     H               ;HL=1000H
00FF 364F                   MVI     M,4FH           ;SCREEN PARAMETER BYTE 1
0101 3658                   MVI     M,58H           ;SCREEN PARAMETER BYTE 2
0103 3689                   MVI     M,89H           ;SCREEN PARAMETER BYTE 3
0105 36DD                   MVI     M,0DDH          ;SCREEN PARAMETER BYTE 4
0107 23                     INX     H               ;HL=1001H
0108 CD8803                 CALL    LDCUR           ;LOAD THE CURSOR
010B 36E0                   MVI     M,0E0H          ;PRESET COUNTERS
010D 3623                   MVI     M,23H           ;START DISPLAY
                    ;
                    ;THIS ROUTINE READS BOTH THE KEYBOARD AND THE USART
                    ;AND TAKES PROPER ACTION DEPENDING ON HOW THE LINE-LOCAL
                    ;SWITCH IS SET
                    ;
010F 3E18           SETUP:  MVI     A,18H           ;SET MASK
0111 30                     SIM                     ;LOAD MASK
0112 FB                     EI                      ;ENABLE INTERRUPTS
                    ;
                    ;READ THE USART
                    ;
0113 20             RXRDY:  RIM                     ;GET LINE LOCAL
0114 E680                   ANI     80H             ;IS IT ON OR OFF?
0116 C22101                 JNZ     KEYINP          ;LEAVE IF IT IS ON
0119 3A01A0                 LDA     USTF            ;READ 8251 FLAGS
011C E602                   ANI     02H             ;LOOK AT RXRDY
011E C25C01                 JNZ     OK7             ;IF HAVE CHARACTER GO TO WORK
0121 3AEA0F         KEYINP: LDA     KEYDWN          ;GET KEYBOARD CHARACTER
0124 E680                   ANI     80H             ;IS IT THERE
0126 C23101                 JNZ     KEYS            ;IF KEY IS PUSHED LEAVE
0129 3E00                   MVI     A,00H           ;ZERO A
012B 32ED0F                 STA     KEYOK           ;CLEAR KEYOK
012E C31301                 JMP     RXRDY           ;LOOP AGAIN
0131 3AED0F         KEYS:   LDA     KEYOK           ;WAS KEY DOWN
0134 4F                     MOV     C,A             ;SAVE A IN C
0135 3AEB0F                 LDA     KBCHR           ;GET KEYBOARD CHARACTER
0138 B9                     CMP     C               ;IS IT THE SAME AS KEYOK
0139 CA1301                 JZ      RXRDY           ;IF SAME LOOP AGAIN
013C 32ED0F                 STA     KEYOK           ;IF NOT SAVE IT
013F 32E70F                 STA     USCHR           ;SAVE IT
0142 20                     RIM                     ;GET LINE LOCAL
0143 E680                   ANI     80H             ;WHICH WAY
0145 CA4B01                 JZ      TRANS           ;LEAVE IF LINE
0148 C34E02                 JMP     CHREC           ;TIME TO DO SOME WORK
014B 3A01A0         TRANS:  LDA     USTF            ;GET USART FLAGS
014E E601                   ANI     01H             ;READY TO TRANSMIT?
0150 CA4B01                 JZ      TRANS           ;LOOP IF NOT READY
0153 3AE70F                 LDA     USCHR           ;GET CHARACTER
0156 3200A0                 STA     USTD            ;PUT IN USART
0159 C30F01                 JMP     SETUP           ;LEAVE
015C 3A00A0         OK7:    LDA     USTD            ;READ USART
015F E67F                   ANI     07FH            ;STRIP MSB
0161 32E70F                 STA     USCHR           ;PUT IT IN MEMORY
0164 C34E02                 JMP     CHREC           ;LEAVE
                    ;
                    ;THIS ROUTINE CHECKS THE BAUD RATE SWITCHES, RESETS THE
                    ;SCREEN POINTERS AND READS AND LOOKS UP THE KEYBOARD.
                    ;
0167 F5             FRAME:  PUSH    PSW             ;SAVE A AND FLAGS
0168 E5                     PUSH    H               ;SAVE H AND L
0169 D5                     PUSH    D               ;SAVE D AND E
016A C5                     PUSH    B               ;SAVE B AND C
016B 3A0114                 LDA     INT75           ;READ 8275 TO CLEAR INTERRUPT
                    ;
                    ;SET UP THE POINTERS
                    ;
016E 2AE30F                 LHLD    TOPAD           ;LOAD TOP IN H AND L
0171 22E80F                 SHLD    CURAD           ;STORE TOP IN CURRENT ADDRESS
                    ;
                    ;SET UP BAUD RATE
                    ;
0174 3A0218                 LDA     PORTC           ;READ BAUD RATE SWITCHES
0177 E60F                   ANI     0FH             ;STRIP OFF 4 MSB'S
0179 47                     MOV     B,A             ;SAVE IN B
017A 3AEC0F                 LDA     BAUD            ;GET BAUD RATE
017D B8                     CMP     B               ;SEE IF SAME AS B
017E C4DC00                 CNZ     STBAUD          ;IF NOT SAME DO SOMETHING
                    ;
                    ;READ KEYBOARD
                    ;
0181 3AEA0F                 LDA     KEYDWN          ;SEE IF A KEY IS DOWN
0184 E640                   ANI     40H             ;SET THE FLAGS
0186 C2C201                 JNZ     KYDOWN          ;IF KEY IS DOWN JUMP AROUND
0189 CD8F01                 CALL    RDKB            ;GO READ THE KEYBOARD
018C C38F00                 JMP     BYPASS          ;LEAVE
```

```
018F 21EF0F      RDKB:   LXI   H,SHCON     ;POINT HL AT KEYBOARD RAM
0192 3A0218              LDA   PORTC      ;GET CONTROL AND SHIFT
0195 77                  MOV   M,A        ;SAVE IN MEMORY
0196 3EFE                MVI   A,0FEH     ;SET UP A
0198 320018      LOOPK:  STA   PORTA      ;OUTPUT A
019B 47                  MOV   B,A        ;SAVE A IN B
019C 3A0118              LDA   PORTB      ;READ KEYBOARD
019F 2F                  CMA              ;INVERT A
01A0 B7                  ORA   A          ;SET THE FLAGS
01A1 C2AF01              JNZ   SAVKEY     ;LEAVE IF KEY IS DOWN
01A4 78                  MOV   A,B        ;GET SCAN LINE BACK
01A5 07                  RLC              ;ROTATE IT OVER ONE
01A6 DA9801              JC    LOOPK      ;DO IT AGAIN
01A9 3E00                MVI   A,00H      ;ZERO A
01AB 32EA0F              STA   KEYDWN     ;SAVE KEY DOWN
01AE C9                  RET              ;LEAVE
01AF 23          SAVKEY: INX   H          ;POINT AT RETURN LINE
01B0 2F                  CMA              ;PUT A BACK
01B1 77                  MOV   M,A        ;SAVE RETURN LINE IN MEMORY
01B2 23                  INX   H          ;POINT H AT SCAN LINE
01B3 70                  MOV   M,B        ;SAVE SCAN LINE IN MEMORY
01B4 3E40                MVI   A,40H      ;SET A
01B6 32EA0F              STA   KEYDWN     ;SAVE KEY DOWN
01B9 C9                  RET              ;LEAVE
01BA 3E00        KYCHNG: MVI   A,00H      ;ZERO 0
01BC 32EA0F              STA   KEYDWN     ;RESET KEY DOWN
01BF C38F00              JMP   BYPASS     ;LEAVE
01C2 21F10F      KYDOWN: LXI   H,SCNLIN   ;GET SCAN LINE
01C5 7E                  MOV   A,M        ;PUT SCAN LINE IN A
01C6 320018              STA   PORTA      ;OUTPUT SCAN LINE TO PORT A
01C9 2B                  DCX   H          ;POINT AT RETURN LINE
01CA 3A0118              LDA   PORTB      ;GET RETURN LINES
01CD B6                  ORA   M          ;ARE THEY THE SAME?
01CE 2F                  CMA              ;INVERT A
01CF B7                  ORA   A          ;SET FLAGS
01D0 CABA01              JZ    KYCHNG     ;IF DIFFERENT KEY HAS CHANGED
01D3 3AEA0F              LDA   KEYDWN     ;GET KEY DOWN
01D6 E601                ANI   01H        ;HAS THIS BEEN DONE BEFORE?
01D8 C28F00              JNZ   BYPASS     ;LEAVE IF IT HAS
01DB 3A0118              LDA   PORTB      ;GET RETURN LINE
01DE 06FF                MVI   B,0FFH     ;GET READY TO ZERO B
01E0 04          UP:     INR   B          ;ZERO B
01E1 0F                  RRC              ;ROTATE A
01E2 DAE001              JC    UP         ;DO IT AGAIN
01E5 23                  INX   H          ;POINT H AT SCAN LINES
01E6 7E                  MOV   A,M        ;GET SCAN LINES
01E7 0EFF                MVI   C,0FFH     ;GET READY TO LOOP
01E9 0C          UP1:    INR   C          ;START C COUNTING
01EA 0F                  RRC              ;ROTATE A
01EB DAE901              JC    UP1        ;JUMP TO LOOP
01EE 78                  MOV   A,B        ;GET RETURN LINES
01EF 07                  RLC              ;MOVE OVER ONCE
01F0 07                  RLC              ;MOVE OVER TWICE
01F1 07                  RLC              ;MOVE OVER THREE TIMES
01F2 B1                  ORA   C          ;OR SCAN AND RETURN LINES
01F3 47                  MOV   B,A        ;SAVE A IN B
01F4 3A0218              LDA   PORTC      ;GET SHIFT CONTROL
01F7 E640                ANI   40H        ;IS CONTROL SET
01F9 4F                  MOV   C,A        ;SAVE A IN C
01FA 3AEF0F              LDA   SHCON      ;GET SHIFT CONTROL
01FD 57                  MOV   D,A        ;SAVE A IN D
01FE E640                ANI   40H        ;STRIP CONTROL
0200 B1                  ORA   C          ;SET BIT
0201 CA3E02              JZ    CNTDWN     ;IF SET LEAVE
0204 3A0218              LDA   PORTC      ;READ IT AGAIN
0207 E620                ANI   20H        ;STRIP SHIFT
0209 4F                  MOV   C,A        ;SAVE A
020A 7A                  MOV   A,D        ;GET SHIFT CONTROL
020B E620                ANI   20H        ;STRIP CONTROL
020D B1                  ORA   C          ;ARE THEY THE SAME?
020E CA4702              JZ    SHDWN      ;IF SET LEAVE
0211 58          SCR:    MOV   E,B        ;PUT  TARGET IN E
0212 1600                MVI   D,00H      ;ZERO D
0214 210705              LXI   H,KYLKUP   ;GET LOOKUP TABLE
0217 19                  DAD   D          ;GET OFFSET
0218 7E                  MOV   A,M        ;GET CHARACTER
0219 47                  MOV   B,A        ;PUT CHARACTER IN B
021A 3A0218              LDA   PORTC      ;GET PORTC
021D E610                ANI   10H        ;STRIP BIT
021F CA2E02              JZ    CAPLOC     ;CAPS LOCK
0222 78                  MOV   A,B        ;GET A BACK
0223 32EB0F      STKEY:  STA   KBCHR      ;SAVE CHARACTER
0226 3EC1                MVI   A,0C1H     ;SET A
0228 32EA0F              STA   KEYDWN     ;SAVE KEY DOWN
022B C38F00              JMP   BYPASS     ;LEAVE
                         ;
                         ;IF THE CAP LOCK BUTTON IS PUSHED THIS ROUTINE SEES IF
                         ;THE CHARACTER IS BETWEEN 61H AND 7AH AND IF IT IS THIS
```

```
                        ;ROUTINE ASSUMES THAT THE CHARACTER IS LOWER CASE ASCII
                        ;AND SUBTRACTS 20H, WHICH CONVERTS THE CHARACTER TO
                        ;UPPER CASE ASCII
                        ;
022E 78                 CAPLOC: MOV     A,B             ;GET A BACK
022F FE60               CPI     60H             ;HOW BIG IS IT?
0231 DA2302             JC      STKEY           ;LEAVE IF IT'S TOO SMALL
0234 FE7B               CPI     7BH             ;IS IT TOO BIG
0236 D22302             JNC     STKEY           ;LEAVE IF TOO BIG
0239 D620               SUI     20H             ;ADJUST A
023B C32302             JMP     STKEY           ;STORE THE KEY
                        ;
                        ;THE ROUTINES SHDWN AND CNTDWN SET BIT 6 AND 7 RESPECTIVLY
                        ;IN THE ACC.
                        ;
023E 3E80               CNTDWN: MVI     A,80H           ;SET BIT 7 IN A
0240 B0                 ORA     B               ;OR WITH CHARACTER
0241 E6BF               ANI     0BFH            ;MAKE SURE SHIFT IS NOT SET
0243 47                 MOV     B,A             ;PUT IT BACK IN B
0244 C31102             JMP     SCR             ;GO BACK
0247 3E40               SHDWN:  MVI     A,40H           ;SET BIT 6 IN A
0249 B0                 ORA     B               ;OR WITH CHARACTER
024A 47                 MOV     B,A             ;PUT IT BACK IN B
024B C31102             JMP     SCR             ;GO BACK
                        ;
                        ;THIS ROUTINE CHECKS FOR ESCAPE CHARACTERS, LF, CR,
                        ;FF, AND BACK SPACE
                        ;
024E 3AEE0F             CHREC:  LDA     ESCP            ;ESCAPE SET?
0251 FE80               CPI     80H             ;SEE IF IT IS
0253 CA7B02             JZ      ESSQ            ;LEAVE IF IT IS
0256 3AE70F             LDA     USCHR           ;GET CHARACTER
0259 FE0A               CPI     0AH             ;LINE FEED
025B CAF603             JZ      LNFD            ;GO TO LINE FEED
025E FE0C               CPI     0CH             ;FORM FEED
0260 CACA03             JZ      FMFD            ;GO TO FORM FEED
0263 FE0D               CPI     0DH             ;CR
0265 CAAD03             JZ      CGRT            ;DO A CR
0268 FE08               CPI     08H             ;BACK SPACE
026A CA6E03             JZ      LEFT            ;DO A BACK SPACE
026D FE1B               CPI     1BH             ;ESCAPE
026F CAA503             JZ      ESKAP           ;DO AN ESCAPE
0272 B7                 ORA     A               ;CLEAR CARRY
0273 C6E0               ADI     0E0H            ;SEE IF CHARACTER IS PRINTABLE
0275 DA7704             JC      CHRPUT          ;IF PRINTABLE DO IT
0278 C30F01             JMP     SETUP           ;GO BACK AND READ USART AGAIN
                        ;
                        ;THIS ROUTINE RESETS THE ESCAPE LOCATION AND DECODES
                        ;THE CHARACTERS FOLLOWING AN ESCAPE. THE COMMANDS ARE
                        ;COMPATABLE WITH INTELS CREDIT TEXT EDITOR
                        ;
027B 3E00               ESSQ:   MVI     A,00H           ;ZERO A
027D 32EE0F             STA     ESCP            ;RESET  ESCP
0280 3AE70F             LDA     USCHR           ;GET CHARACTER
0283 FE42               CPI     42H             ;DOWN
0285 CAAE02             JZ      DOWN            ;MOVE CURSOR DOWN
0288 FE45               CPI     45H             ;CLEAR SCREEN CHARACTER
028A CACF02             JZ      CLEAR           ;CLEAR THE SCREEN
028D FE4A               CPI     4AH             ;CLEAR REST OF SCREEN
028F CAD502             JZ      CLRST           ;GO CLEAR THE REST OF THE SCREEN
0292 FE4B               CPI     4BH             ;CLEAR LINE CHARACTER
0294 CA2703             JZ      CLRLIN          ;GO CLEAR A LINE
0297 FE41               CPI     41H             ;CURSOR UP CHARACTER
0299 CA3303             JZ      UPCUR           ;MOVE CURSOR UP
029C FE43               CPI     43H             ;CURSOR RIGHT CHARACTER
029E CA4503             JZ      RIGHT           ;MOVE CURSOR TO THE RIGHT
02A1 FE44               CPI     44H             ;CURSOR LEFT CHARACTER
02A3 CA6E03             JZ      LEFT            ;MOVE CURSOR TO THE LEFT
02A6 FE48               CPI     48H             ;HOME CURSOR CHARACTER
02A8 CA9703             JZ      HOME            ;HOME THE CURSOR
02AB C30F01             JMP     SETUP           ;LEAVE
                        ;
                        ;THIS ROUTINE MOVES THE CURSOR DOWN ONE CHARACTER LINE
                        ;
02AE 3AE10F             DOWN:   LDA     CURSY           ;PUT CURSOR Y IN A
02B1 FE18               CPI     CURBOT          ;SEE IF ON BOTTOM OF SCREEN
02B3 CA0F01             JZ      SETUP           ;LEAVE IF ON BOTTOM
02B6 3C                 INR     A               ;INCREMENT Y CURSOR
02B7 32E10F             STA     CURSY           ;SAVE NEW CURSOR
02BA CDB803             CALL    LDCUR           ;LOAD THE CURSOR
02BD CDA504             CALL    CALCU           ;CALCULATE ADDRESS
02C0 7E                 MOV     A,M             ;GET FIRST LOCATION OF THE LINE
02C1 FEF0               CPI     0F0H            ;SEE IF CLEAR SCREEN CHARACTER
02C3 C20F01             JNZ     SETUP           ;LEAVE IF IT IS NOT
02C6 22E50F             SHLD    LOC80           ;SAVE BEGINNING OF THE LINE
02C9 CD1504             CALL    CLLINE          ;CLEAR THE LINE
02CC C30F01             JMP     SETUP           ;LEAVE
                        ;
```

```
                ;THIS ROUTINE CLEARS THE SCREEN.
                ;
02CF CDE403     CLEAR:  CALL    CLSCR           ;GO CLEAR THE SCREEN
02D2 C30F01             JMP     SETUP           ;GO BACK
                ;
                ;THIS ROUTINE CLEARS ALL LINES BENEATH THE LOCATION
                ;OF THE CURSOR.
                ;
02D5 CDA504     CLRST:  CALL    CALCU           ;CALCULATE ADDRESS
02D8 CDCD04             CALL    ADX             ;ADD X POSITION
02DB 01204F             LXI     B,4F20H         ;PUT SPACE AND LAST X IN B AND C
02DE 3AE20F             LDA     CURSX           ;GET X CURSOR
02E1 B8                 CMP     B               ;SEE IF AT END OF LINE
02E2 CAEC02             JZ      OVR1            ;LEAVE IF X IS AT END OF LINE
02E5 3C         LLP:    INR     A               ;MOVE A OVER ONE X POSITION
02E6 23                 INX     H               ;INCREMENT MEMORY POINTER
02E7 71                 MOV     M,C             ;PUT A SPACE IN MEMORY
02E8 B8                 CMP     B               ;SEE IF A = 4FH
02E9 C2E502             JNZ     LLP             ;IF NOT LOOP AGAIN
02EC 01D00F     OVR1:   LXI     B,LAST          ;PUT LAST LINE IN BC
02EF 23                 INX     H               ;POINT HL TO LAST LINE
02F0 78                 MOV     A,B             ;GET B
02F1 BC                 CMP     H               ;SAME AS H?
02F2 C2FD02             JNZ     CONCL           ;LEAVE IF NOT
02F5 79                 MOV     A,C             ;GET C
02F6 BD                 CMP     L               ;SAME AS L?
02F7 C2FD02             JNZ     CONCL           ;LEAVE IF NOT
02FA 210008             LXI     H,TPDIS         ;GET TOP OF DISPLAY
02FD 3AE10F     CONCL:  LDA     CURSY           ;GET Y CURSOR
0300 FE18               CPI     CURBOT          ;IS IT ON THE BOTTOM
0302 CA0F01             JZ      SETUP           ;LEAVE IF IT IS
0305 3C                 INR     A               ;MOVE IT DOWN ONE LINE
0306 47                 MOV     B,A             ;SAVE CURSOR IN B FOR LATER
0307 115000             LXI     D,LNGTH         ;PUT LENGTH OF ONE LINE IN D
030A 36F0       CLOOP:  MVI     M,0F0H          ;PUT EOR IN MEMORY
030C 78                 MOV     A,B             ;GET CURSOR Y
030D FE18               CPI     CURBOT          ;ARE WE ON THE BOTTOM
030F CA0F01             JZ      SETUP           ;LEAVE IF WE ARE
0312 3C                 INR     A               ;MOVE CURSOR DOWN ONE
0313 19                 DAD     D               ;GET NEXT LINE
0314 47                 MOV     B,A             ;SAVE A
0315 7C                 MOV     A,H             ;PUT H IN A
0316 FE0F               CPI     0FH             ;COMPARE TO HIGH LAST
0318 C20A03             JNZ     CLOOP           ;LEAVE IF IT IS NOT
031B 7D                 MOV     A,L             ;PUT L IN A
031C FED0               CPI     0D0H            ;COMPARE TO LOW LAST
031E C20A03             JNZ     CLOOP           ;LEAVE IF IT IS NOT
0321 210008             LXI     H,TPDIS         ;PUT TOP DISPLAY IN H AND L
0324 C30A03             JMP     CLOOP           ;LOOP AGAIN
                ;
                ;THIS ROUTINE CLEARS THE LINE THE CURSOR IS ON.
                ;
0327 CDA504     CLRLIN: CALL    CALCU           ;CALCULATE ADDRESS
032A 22E50F             SHLD    LOC80           ;STORE H AND L TO CLEAR LINE
032D CD1504             CALL    CLLINE          ;CLEAR THE LINE
0330 C30F01             JMP     SETUP           ;GO BACK
                ;
                ;THIS ROUTINE MOVES THE CURSOR UP ONE LINE.
                ;
0333 3AE10F     UPCUR:  LDA     CURSY           ;GET Y CURSOR
0336 FE00               CPI     00H             ;IS IT ZERO
0338 CA0F01             JZ      SETUP           ;IF IT IS LEAVE
033B 3D                 DCR     A               ;MOVE CURSOR UP
033C 32E10F             STA     CURSY           ;SAVE NEW CURSOR
033F CDB803             CALL    LDCUR           ;LOAD THE CURSOR
0342 C30F01             JMP     SETUP           ;LEAVE
                ;
                ;THIS ROUTINE MOVES THE CURSOR ONE LOCATION TO THE RIGHT
                ;
0345 3AE20F     RIGHT:  LDA     CURSX           ;GET X CURSOR
0348 FE4F               CPI     4FH             ;IS IT ALL THE WAY OVER?
034A C26403             JNZ     NTOVER          ;IF NOT JUMP AROUND
034D 3AE10F             LDA     CURSY           ;GET Y CURSOR
0350 FE18               CPI     CURBOT          ;SEE IF ON BOTTOM
0352 CA5903             JZ      GD18            ;IF WE ARE JUMP
0355 3C                 INR     A               ;INCREMENT Y CURSOR
0356 32E10F             STA     CURSY           ;SAVE IT
0359 3E00       GD18:   MVI     A,00H           ;ZERO A
035B 32E20F             STA     CURSX           ;ZERO X CURSOR
035E CDB803             CALL    LDCUR           ;LOAD THE CURSOR
0361 C30F01             JMP     SETUP           ;LEAVE
0364 3C         NTOVER: INR     A               ;INCREMENT X CURSOR
0365 32E20F             STA     CURSX           ;SAVE IT
0368 CDB803             CALL    LDCUR           ;LOAD THE CURSOR
036B C30F01             JMP     SETUP           ;LEAVE
                ;
                ;THIS ROUTINE MOVES THE CURSOR LEFT ONE CHARACTER POSITION
```

```
                    570          ;
036E 3AE20F         571 LEFT:    LDA     CURSX          ;GET X CURSOR
0371 FE00           572           CPI     00H            ;IS IT ALL THE WAY OVER
0373 C28D03         573           JNZ     NOVER          ;IF NOT JUMP AROUND
0376 3AE10F         574           LDA     CURSY          ;GET CURSOR Y
0379 FE00           575           CPI     00H            ;IS IT ZERO?
037B CA0F01         576           JZ      SETUP          ;IF IT IS JUMP
037E 3D             577           DCR     A              ;MOVE CURSOR Y UP
037F 32E10F         578           STA     CURSY          ;SAVE IT
0382 3E4F           579           MVI     A,4FH          ;GET LAST X LOCATION
0384 32E20F         580           STA     CURSX          ;SAVE IT
0387 CDB803         581           CALL    LDCUR          ;LOAD THE CURSOR
038A C30F01         582           JMP     SETUP
038D 3D             583 NOVER:    DCR     A              ;ADJUST X CURSOR
038E 32E20F         584           STA     CURSX          ;SAVE CURSOR X
0391 CDB803         585           CALL    LDCUR          ;LOAD THE CURSOR
0394 C30F01         586           JMP     SETUP          ;LEAVE
                    587          ;
                    588          ;THIS ROUTINE HOMES THE CURSOR.
                    589          ;
0397 3E00           590 HOME:     MVI     A,00H          ;ZERO A
0399 32E20F         591           STA     CURSX          ;ZERO X CURSOR
039C 32E10F         592           STA     CURSY          ;ZERO Y CURSOR
039F CDB803         593           CALL    LDCUR          ;LOAD THE CURSOR
03A2 C30F01         594           JMP     SETUP          ;LEAVE
                    595          ;
                    596          ;THIS ROUTINE SETS THE ESCAPE BIT
                    597          ;
03A5 3E80           598 ESKAP:    MVI     A,80H          ;LOAD A WITH ESCAPE BIT
03A7 32EE0F         599           STA     ESCP           ;SET ESCAPE LOCATION
03AA C30F01         600           JMP     SETUP          ;GO BACK AND READ USART
                    601          ;
                    602          ;THIS ROUTINE DOES A CR
                    603          ;
03AD 3E00           604 CGRT:     MVI     A,00H          ;ZERO A
03AF 32E20F         605           STA     CURSX          ;ZERO  CURSOR X
03B2 CDB803         606           CALL    LDCUR          ;LOAD CURSOR INTO 8275
03B5 C30F01         607           JMP     SETUP          ;POLL USART AGAIN
                    608          ;
                    609          ;THIS ROUTINE LOADS THE CURSOR
                    610          ;
03B8 3E80           611 LDCUR:    MVI     A,80H          ;PUT 80H INTO A
03BA 320110         612           STA     CRTS           ;LOAD CURSOR INTO 8275
03BD 3AE20F         613           LDA     CURSX          ;GET CURSOR X
03C0 320010         614           STA     CRTM           ;PUT IT IN 8275
03C3 3AE10F         615           LDA     CURSY          ;GET CURSOR Y
03C6 320010         616           STA     CRTM           ;PUT IT IN 8275
03C9 C9             617           RET
                    618          ;
                    619          ;THIS ROUTINE DOES A FORM FEED
                    620          ;
03CA CDE403         621 FMFD:     CALL    CLSCR          ;CALL CLEAR SCREEN
03CD 210008         622           LXI     H,TPDIS        ;PUT TOP DISPLAY IN HL
03D0 22E50F         623           SHLD    LOC80          ;PUT IT IN LOC80
03D3 CD1504         624           CALL    CLLINE         ;CLEAR TOP LINE
03D6 3E00           625           MVI     A,00H          ;ZERO A
03D8 32E20F         626           STA     CURSX          ;ZERO CURSOR X
03DB 32E10F         627           STA     CURSY          ;ZERO CURSOR Y
03DE CDB803         628           CALL    LDCUR          ;LOAD THE CURSOR
03E1 C30F01         629           JMP     SETUP          ;BACK TO USART
                    630          ;
                    631          ;THIS ROUTINE CLEARS THE SCREEN BY WRITING END OF ROW
                    632          ;CHARACTERS INTO THE FIRST LOCATION OF ALL LINES ON
                    633          ;THE SCREEN.
                    634          ;
03E4 3EF0           635 CLSCR:    MVI     A,0F0H         ;PUT EOR CHARACTER IN A
03E6 0618           636           MVI     B,CURBOT       ;LOAD B WITH MAX Y
03E8 04             637           INR     B              ;GO TO MAX PLUS ONE
03E9 210008         638           LXI     H,TPDIS        ;LOAD H AND L WITH TOP OF RAM
03EC 115000         639           LXI     D,LNGTH        ;MOVE 50H = 80D INTO D AND E
03EF 77             640 LOADX:    MOV     M,A            ;MOVE EOR INTO MEMORY
03F0 19             641           DAD     D              ;CHANGE POINTER BY 80D
03F1 05             642           DCR     B              ;COUNT THE LOOPS
03F2 C2EF03         643           JNZ     LOADX          ;CONTINUE IF NOT ZERO
03F5 C9             644           RET                    ;GO BACK
                    645          ;
                    646          ;THIS ROUTINE DOES A LINE FEED
                    647          ;
03F6 CDFC03         648 LNFD:     CALL    LNFD1          ;CALL ROUTINE
03F9 C30F01         649           JMP     SETUP          ;POLL FLAGS
                    650          ;
                    651          ;LINE FEED
                    652          ;
03FC 3AE10F         653 LNFD1:    LDA     CURSY          ;GET Y LOCATION OF CURSOR
03FF FE18           654           CPI     CURBOT         ;SEE IF AT BOTTOM OF SCREEN
0401 CA5304         655           JZ      ONBOT          ;IF WE ARE, LEAVE
0404 3C             656           INR     A              ;INCREMENT A
0405 32E10F         657           STA     CURSY          ;SAVE NEW CURSOR
```

```
0408 CDA504          CALL    CALCU           ;CALCULATE ADDRESS
040B 22E50F          SHLD    LOC80           ;SAVE TO CLEAR LINE
040E CD1504          CALL    CLLINE          ;CLEAR THE LINE
0411 CD8803          CALL    LDCUR           ;LOAD THE CURSOR
0414 C9              RET                     ;LEAVE
                     ;
                     ;THIS ROUTINE CLEARS THE LINE WHOSE FIRST ADDRESS
                     ;IS IN LOC80. PUSH INSTRUCTIONS ARE USED TO RAPIDLY
                     ;CLEAR THE LINE
                     ;
0415 F3      CLLINE: DI                      ;NO INTERRUPTS HERE
0416 2AE50F          LHLD    LOC80           ;GET LOC80
0419 115000          LXI     D,LNGTH         ;GET OFFSET
041C 19              DAD     D               ;ADD OFFSET
041D EB              XCHG                    ;PUT START IN DE
041E 210000          LXI     H,0000H         ;ZERO HL
0421 39              DAD     SP              ;GET STACK
0422 EB              XCHG                    ;PUT STACK IN DE
0423 F9              SPHL                    ;PUT START IN SP
0424 212020          LXI     H,2020H         ;PUT SPACES IN HL
                     ;
                     ;NOW DO 40 PUSH INSTRUCTIONS TO CLEAR THE LINE
                     ;
                     REPT (LNGTH/2)
                     PUSH    H
                     ENDM
0427 E5              PUSH    H
0428 E5              PUSH    H
0429 E5              PUSH    H
042A E5              PUSH    H
042B E5              PUSH    H
042C E5              PUSH    H
042D E5              PUSH    H
042E E5              PUSH    H
042F E5              PUSH    H
0430 E5              PUSH    H
0431 E5              PUSH    H
0432 E5              PUSH    H
0433 E5              PUSH    H
0434 E5              PUSH    H
0435 E5              PUSH    H
0436 E5              PUSH    H
0437 E5              PUSH    H
0438 E5              PUSH    H
0439 E5              PUSH    H
043A E5              PUSH    H
043B E5              PUSH    H
043C E5              PUSH    H
043D E5              PUSH    H
043E E5              PUSH    H
043F E5              PUSH    H
0440 E5              PUSH    H
0441 E5              PUSH    H
0442 E5              PUSH    H
0443 E5              PUSH    H
0444 E5              PUSH    H
0445 E5              PUSH    H
0446 E5              PUSH    H
0447 E5              PUSH    H
0448 E5              PUSH    H
0449 E5              PUSH    H
044A E5              PUSH    H
044B E5              PUSH    H
044C E5              PUSH    H
044D E5              PUSH    H
044E E5              PUSH    H
044F EB              XCHG                    ;PUT STACK IN HL
0450 F9              SPHL                    ;PUT IT BACK IN SP
0451 FB              EI                      ;ENABLE INTERRUPTS
0452 C9              RET                     ;GO BACK
                     ;
                     ;IF CURSOR IS ON THE BOTTOM OF THE SCREEN THIS ROUTINE
                     ;IS USED TO IMPLEMENT THE LINE FEED
                     ;
0453 2AE30F  ONBOT:  LHLD    TOPAD           ;GET TOP ADDRESS
0456 22E50F          SHLD    LOC80           ;SAVE IT IN LOC80
0459 115000          LXI     D,LNGTH         ;LINE LENGTH
045C 19              DAD     D               ;ADD HL + DE
045D 01D00F          LXI     B,LAST          ;GET BOTTOM LINE
0460 7C              MOV     A,H             ;GET H
0461 B8              CMP     B               ;SAME AS B
0462 C26D04          JNZ     ARND            ;LEAVE IF NOT SAME
0465 7D              MOV     A,L             ;GET L
0466 B9              CMP     C               ;SAME AS C
0467 C26D04          JNZ     ARND            ;LEAVE IF NOT SAME
046A 210008          LXI     H,TPDIS         ;LOAD HL WITH TOP OF DISPLAY
046D 22E30F  ARND:   SHLD    TOPAD           ;SAVE NEW TOP ADDRESS
```

```
0470 CD1504          745          CALL    CLLINE          ;CLEAR LINE
0473 CDB803          746          CALL    LDCUR           ;LOAD THE CURSOR
0476 C9              747          RET
                     748          ;
                     749          ;THIS ROUTINE PUTS A CHARACTER ON THE SCREEN AND
                     750          ;INCREMENTS THE X CURSOR POSITION. A LINE FEED IS
                     751          ;INSERTED IF THE INCREMENTED CURSOR EQUALS 81D
                     752          ;
0477 CDA504          753 CHRPUT:  CALL    CALCU           ;CALCULATE SCREEN POSITION
047A 7E              754          MOV     A,M             ;GET FIRST CHARACTER
047B FEF0            755          CPI     0F0H            ;IS IT A CLEAR LINE
047D 22E50F          756          SHLD    LOC80           ;SAVE LINE TO CLEAR
0480 CC1504          757          CZ      CLLINE          ;CLEAR LINE
0483 2AE50F          758          LHLD    LOC80           ;GET LINE
0486 CDCD04          759          CALL    ADX             ;ADD CURSOR X
0489 3AE70F          760          LDA     USCHR           ;GET CHARACTER
048C 77              761          MOV     M,A             ;PUT IT ON SCREEN
048D 3AE20F          762          LDA     CURSX           ;GET CURSOR X
0490 3C              763          INR     A               ;INCREMENT CURSOR X
0491 FE50            764          CPI     LNGTH           ;HAS IT GONE TOO FAR?
0493 C29C04          765          JNZ     OK1             ;IF NOT GOOD
0496 CDFC03          766          CALL    LNFD1           ;DO A LINE FEED
0499 C3AD03          767          JMP     CGRT            ;DO A CR
049C 32E20F          768 OK1:     STA     CURSX           ;SAVE CURSOR
049F CDB803          769          CALL    LDCUR           ;LOAD THE CURSOR
04A2 C30F01          770          JMP     SETUP           ;LEAVE
                     771          ;
                     772          ;THIS ROUTINE TAKES THE TOP ADDRESS AND THE Y CURSOR
                     773          ;LOCATION AND CALCULATES THE ADDRESS OF THE LINE
                     774          ;THAT THE CURSOR IS ON. THE RESULT IS RETURNED IN H
                     775          ;AND L AND ALL REGISTERS ARE USED.
                     776          ;
04A5 21D504          777 CALCU:   LXI     H,LINTAB        ;GET LINE TABLE INTO H AND L
04A8 3AE10F          778          LDA     CURSY           ;GET CURSOR INTO A
04AB 07              779          RLC                     ;SET UP A FOR LOOKUP TABLE
04AC 0600            780          MVI     B,00H           ;ZERO B
04AE 4F              781          MOV     C,A             ;PUT CURSOR INTO A
04AF 09              782          DAD     B               ;ADD LINE TABLE TO Y CURSOR
04B0 7E              783          MOV     A,M             ;PUT LOW LINE TABLE INTO A
04B1 4F              784          MOV     C,A             ;PUT LOW LINE TABLE INTO C
04B2 23              785          INX     H               ;CHANGE MEMORY POINTER
04B3 7E              786          MOV     A,M             ;PUT HIGH LINE TABLE INTO A
04B4 47              787          MOV     B,A             ;PUT HIGH LINE TABLE INTO B
04B5 2100F8          788          LXI     H,0F800H        ;TWOS COMPLEMENT SCREEN LOCATION
04B8 09              789          DAD     B               ;SUBTRACT OFFSET
04B9 EB              790          XCHG                    ;SAVE HL IN DE
04BA 2AE30F          791          LHLD    TOPAD           ;GET TOP ADDRESS IN H AND L
04BD 19              792          DAD     D               ;GET DISPLACED ADDRESS
04BE EB              793          XCHG                    ;SAVE IT IN D
04BF 2130F0          794          LXI     H,0F030H        ;TWOS COMPLEMENT SCREEN LOCATION
04C2 19              795          DAD     D               ;SEE IF WE ARE OFF THE SCREEN
04C3 DAC804          796          JC      FIX             ;IF WE ARE FIX IT
04C6 EB              797          XCHG                    ;GET DISPLACED ADDRESS BACK
04C7 C9              798          RET                     ;GO BACK
04C8 2130F8          799 FIX:     LXI     H,0F830H        ;SCREEN BOUNDRY
04CB 19              800          DAD     D               ;ADJUST SCREEN
04CC C9              801          RET                     ;GO BACK
                     802          ;
                     803          ;THIS ROUTINE ADDS THE X CURSOR LOCATION TO THE ADDRESS
                     804          ;THAT IS IN THE H AND L REGISTERS AND RETURNS THE RESULT
                     805          ;IN H AND L
                     806          ;
04CD 3AE20F          807 ADX:     LDA     CURSX           ;GET CURSOR
04D0 0600            808          MVI     B,00H           ;ZERO B
04D2 4F              809          MOV     C,A             ;PUT CURSOR X IN C
04D3 09              810          DAD     B               ;ADD CURSOR X TO H AND L
04D4 C9              811          RET                     ;LEAVE
                     812          ;
                     813          ;THIS TABLE CONTAINS THE OFFSET ADDRESSES FOR EACH
                     814          ;OF THE 25 DISPLAYED LINES.
                     815          ;
0000                 816 LINTAB:  LINNUM SET 0
                     817          REPT (CURBOT+1)
                     818          DW      TPDIS+(LNGTH*LINNUM)
                     819          LINNUM SET (LINNUM+1)
                     820          ENDM
04D5 0008            821+         DW      TPDIS+(LNGTH*LINNUM)
0001                 822+         LINNUM SET (LINNUM+1)
04D7 5008            823+         DW      TPDIS+(LNGTH*LINNUM)
0002                 824+         LINNUM SET (LINNUM+1)
04D9 A008            825+         DW      TPDIS+(LNGTH*LINNUM)
0003                 826+         LINNUM SET (LINNUM+1)
04DB F008            827+         DW      TPDIS+(LNGTH*LINNUM)
0004                 828+         LINNUM SET (LINNUM+1)
04DD 4009            829+         DW      TPDIS+(LNGTH*LINNUM)
0005                 830+         LINNUM SET (LINNUM+1)
04DF 9009            831+         DW      TPDIS+(LNGTH*LINNUM)
```

```
0006                 832+        LINNUM SET (LINNUM+1)
04E1 E009            833+        DW      TPDIS+(LNGTH*LINNUM)
0007                 834+        LINNUM SET (LINNUM+1)
04E3 300A            835+        DW      TPDIS+(LNGTH*LINNUM)
0008                 836+        LINNUM SET (LINNUM+1)
04E5 800A            837+        DW      TPDIS+(LNGTH*LINNUM)
0009                 838+        LINNUM SET (LINNUM+1)
04E7 D00A            839+        DW      TPDIS+(LNGTH*LINNUM)
000A                 840+        LINNUM SET (LINNUM+1)
04E9 200B            841+        DW      TPDIS+(LNGTH*LINNUM)
000B                 842+        LINNUM SET (LINNUM+1)
04EB 700B            843+        DW      TPDIS+(LNGTH*LINNUM)
000C                 844+        LINNUM SET (LINNUM+1)
04ED C00B            845+        DW      TPDIS+(LNGTH*LINNUM)
000D                 846+        LINNUM SET (LINNUM+1)
04EF 100C            847+        DW      TPDIS+(LNGTH*LINNUM)
000E                 848+        LINNUM SET (LINNUM+1)
04F1 600C            849+        DW      TPDIS+(LNGTH*LINNUM)
000F                 850+        LINNUM SET (LINNUM+1)
04F3 B00C            851+        DW      TPDIS+(LNGTH*LINNUM)
0010                 852+        LINNUM SET (LINNUM+1)
04F5 000D            853+        DW      TPDIS+(LNGTH*LINNUM)
0011                 854+        LINNUM SET (LINNUM+1)
04F7 500D            855+        DW      TPDIS+(LNGTH*LINNUM)
0012                 856+        LINNUM SET (LINNUM+1)
04F9 A00D            857+        DW      TPDIS+(LNGTH*LINNUM)
0013                 858+        LINNUM SET (LINNUM+1)
04FB F00D            859+        DW      TPDIS+(LNGTH*LINNUM)
0014                 860+        LINNUM SET (LINNUM+1)
04FD 400E            861+        DW      TPDIS+(LNGTH*LINNUM)
0015                 862+        LINNUM SET (LINNUM+1)
04FF 900E            863+        DW      TPDIS+(LNGTH*LINNUM)
0016                 864+        LINNUM SET (LINNUM+1)
0501 E00E            865+        DW      TPDIS+(LNGTH*LINNUM)
0017                 866+        LINNUM SET (LINNUM+1)
0503 300F            867+        DW      TPDIS+(LNGTH*LINNUM)
0018                 868+        LINNUM SET (LINNUM+1)
0505 800F            869+        DW      TPDIS+(LNGTH*LINNUM)
0019                 870+        LINNUM SET (LINNUM+1)
                     871         ;
                     872         ;KEYBOARD LOOKUP TABLE
                     873         ;THIS TABLE CONTAINS ALL THE ASCII CHARACTERS
                     874         ;THAT ARE TRANSMITTED BY THE TERMINAL
                     875         ;THE CHARACTERS ARE ORGANIZED SO THAT BITS 0,1 AND 2
                     876         ;ARE THE SCAN LINES, BITS 3,4 AND 5 ARE THE RETURN LINES
                     877         ;BIT 6 IS SHIFT AND BIT 7 IS CONTROL
                     878         ;
0507 38              879 KYLKUP: DB      38H,39H         ;8 AND 9
0508 39
0509 30              880         DB      30H,2DH         ;0 AND -
050A 2D
050B 3D              881         DB      3DH,5CH         ;= AND \
050C 5C
050D 08              882         DB      08H,00H         ;BS AND BREAK
050E 00
050F 75              883         DB      75H,69H         ;LOWER CASE U AND I
0510 69
0511 6F              884         DB      6FH,70H         ;LOWER CASE O AND P
0512 70
0513 5B              885         DB      5BH,5CH         ;[ AND \
0514 5C
0515 0A              886         DB      0AH,7FH         ;LF AND DELETE
0516 7F
0517 6A              887         DB      6AH,6BH         ;LOWER CASE J AND K
0518 6B
0519 6C              888         DB      6CH,3BH         ;LOWER CASE L AND ;
051A 3B
051B 27              889         DB      27H,00H         ;' AND NOTHING
051C 00
051D 0D              890         DB      0DH,37H         ;CR AND 7
051E 37
051F 6D              891         DB      6DH,2CH         ;LOWER CASE M AND COMMA
0520 2C
0521 2E              892         DB      2EH,2FH         ;PERIOD AND SLASH
0522 2F
0523 00              893         DB      00H,00H         ;BLANK AND NOTHING
0524 00
0525 00              894         DB      00H,00H         ;NOTHING AND NOTHING
0526 00
0527 00              895         DB      00H,61H         ;NOTHING AND LOWER CASE A
0528 61
0529 7A              896         DB      7AH,78H         ;LOWER CASE Z AND X
052A 78
052B 63              897         DB      63H,76H         ;LOWER CASE C AND V
052C 76
052D 62              898         DB      62H,6EH         ;LOWER CASE B AND N
052E 6E
```

```
052F 79      899     DB      79H,00H     ;LOWER CASE Y AND NOTHING
0530 00
0531 00      900     DB      00H,20H     ;NOTHING AND SPACE
0532 20
0533 64      901     DB      64H,66H     ;LOWER CASE D AND F
0534 66
0535 67      902     DB      67H,68H     ;LOWER CASE G AND H
0536 68
0537 00      903     DB      00H,71H     ;TAB AND LOWER CASE Q
0538 71
0539 77      904     DB      77H,73H     ;LOWER CASE W AND S
053A 73
053B 65      905     DB      65H,72H     ;LOWER CASE E AND R
053C 72
053D 74      906     DB      74H,00H     ;LOWER CASE T AND NOTHING
053E 00
053F 1B      907     DB      1BH,31H     ;ESCAPE AND 1
0540 31
0541 32      908     DB      32H,33H     ; 2 AND 3
0542 33
0543 34      909     DB      34H,35H     ; 4 AND 5
0544 35
0545 36      910     DB      36H,00H     ; 6 AND NOTHING
0546 00
0547 2A      911     DB      2AH,28H     ;* AND )
0548 28
0549 29      912     DB      29H,5FH     ;( AND -
054A 5F
054B 2B      913     DB      2BH,00H     ;+ AND NOTHING
054C 00
054D 08      914     DB      08H,00H     ;BS AND BREAK
054E 00
054F 55      915     DB      55H,49H     ;U AND I
0550 49
0551 4F      916     DB      4FH,50H     ;O AND P
0552 50
0553 5D      917     DB      5DH,00H     ;] AND NO CHARACTER
0554 00
0555 0A      918     DB      0AH,7FH     ;LF AND DELETE
0556 7F
0557 4A      919     DB      4AH,4BH     ;J AND K
0558 4B
0559 4C      920     DB      4CH,3AH     ;L AND :
055A 3A
055B 22      921     DB      22H,00H     ;" AND NO CHARACTER
055C 00
055D 0D      922     DB      0DH,26H     ;CR AND &
055E 26
055F 4D      923     DB      4DH,3CH     ;M AND <
0560 3C
0561 3E      924     DB      3EH,3FH     ;> AND ?
0562 3F
0563 00      925     DB      00H,00H     ;BLANK AND NOTHING
0564 00
0565 00      926     DB      00H,00H     ;NOTHING AND NOTHING
0566 00
0567 00      927     DB      00H,41H     ;NOTHING AND A
0568 41
0569 5A      928     DB      5AH,58H     ;Z AND X
056A 58
056B 43      929     DB      43H,56H     ;C AND V
056C 56
056D 42      930     DB      42H,4EH     ;B AND N
056E 4E
056F 59      931     DB      59H,00H     ;Y AND NOTHING
0570 00
0571 00      932     DB      00H,20H     ;NO CHARACTER AND SPACE
0572 20
0573 44      933     DB      44H,46H     ;D AND F
0574 46
0575 47      934     DB      47H,48H     ;G AND H
0576 48
0577 00      935     DB      00H,51H     ;TAB AND Q
0578 51
0579 57      936     DB      57H,53H     ;W AND S
057A 53
057B 45      937     DB      45H,52H     ;E AND R
057C 52
057D 54      938     DB      54H,00H     ;T AND NO CONNECTION
057E 00
057F 1B      939     DB      1BH,21H     ;ESCAPE AND !
0580 21
0581 40      940     DB      40H,23H     ;@ AND #
0582 23
0583 24      941     DB      24H,25H     ;$ AND %
0584 25
0585 5E      942     DB      5EH,00H     ;^ AND NO CONNECTION
```

```
0586 00
                    943          ;
                    944          ;THIS IS WHERE THE CONTROL CHARACTERS ARE LOOKED UP
                    945          ;
0587 00             946          DB      00H,00H          ;NOTHING
0588 00
0589 00             947          DB      00H,00H          ;NOTHING
058A 00
058B 00             948          DB      00H,00H          ;NOTHING
058C 00
058D 00             949          DB      00H,00H          ;NOTHING
058E 00
058F 15             950          DB      15H,09H          ;CONTROL U AND I
0590 09
0591 0F             951          DB      0FH,10H          ;CONTROL O AND P
0592 10
0593 0B             952          DB      0BH,0CH          ;CONTROL [ AND \
0594 0C
0595 0A             953          DB      0AH,7FH          ;LF AND DELETE
0596 7F
0597 0A             954          DB      0AH,0BH          ;CONTROL J AND K
0598 0B
0599 0C             955          DB      0CH,00H          ;CONTROL L AND NOTHING
059A 00
059B 00             956          DB      00H,00H          ;NOTHING
059C 00
059D 0D             957          DB      0DH,00H          ;CR AND NOTHING
059E 00
059F 0D             958          DB      0DH,00H          ;CONTROL M AND COMMA
05A0 00
05A1 00             959          DB      00H,00H          ;NOTHING
05A2 00
05A3 00             960          DB      00H,00H          ;NOTHING
05A4 00
05A5 00             961          DB      00H,00H          ;NOTHING AND NOTHING
05A6 00
05A7 1A             962          DB      1AH,18H          ;CONTROL Z AND X
05A8 18
05A9 03             963          DB      03H,16H          ;CONTROL C AND V
05AA 16
05AB 02             964          DB      02H,0EH          ;CONTROL B AND N
05AC 0E
05AD 19             965          DB      19H,00H          ;CONTROL Y AND NOTHING
05AE 00
05AF 00             966          DB      00H,20H          ;NOTHING AND SPACE
05B0 20
05B1 04             967          DB      04H,06H          ;CONTROL D AND F
05B2 06
05B3 07             968          DB      07H,08H          ;CONTROL G AND H
05B4 08
05B5 00             969          DB      00H,11H          ;NOTHING AND CONTROL Q
05B6 11
05B7 17             970          DB      17H,13H          ;CONTROL W AND S
05B8 13
05B9 06             971          DB      06H,12H          ;CONTROL E AND R
05BA 12
05BB 14             972          DB      14H,00H          ;CONTROL W AND NOTHING
05BC 00
05BD 1B             973          DB      1BH,1DH          ;ESCAPE AND HOME(CREDIT)
05BE 1D
05BF 1E             974          DB      1EH,1CH          ;CURSOR UP AND DOWN(CREDIT)
05C0 1C
05C1 14             975          DB      14H,1FH          ;CURSOR RIGHT AND LEFT(CREDIT)
05C2 1F
05C3 00             976          DB      00H,00H          ;NOTHING
05C4 00
                    977          ;
                    978          ;LOOK UP TABLE FOR 8253 BAUD RATE GENERATOR
                    979          ;
05C5 00             980 BDLK:    DB      00H,05H,69H,03H ;75 AND 110 BAUD
05C6 05
05C7 69
05C8 03
05C9 80             981          DB      80H,02H,40H,01H ;150 AND 300 BAUD
05CA 02
05CB 40
05CC 01
05CD A0             982          DB      0A0H,00H         ;600 BAUD
05CE 00
05CF 50             983          DB      50H,00H          ;1200 BAUD
05D0 00
05D1 28             984          DB      28H,00H          ;2400 BAUD
05D2 00
05D3 14             985          DB      14H,00H          ;4800 BAUD
05D4 00
05D5 0A             986          DB      0AH,00H          ;9600 BAUD
05D6 00
```

```
                 987          ;
                 988          ;DATA AREA
                 989          ;
0FE1             990          ORG     0FE1H
0001             991 CURSY:   DS      1
0001             992 CURSX:   DS      1
0002             993 TOPAD:   DS      2
0002             994 LOC80:   DS      2
0001             995 USCHR:   DS      1
0002             996 CURAD:   DS      2
0001             997 KEYDWN:  DS      1
0001             998 KBCHR:   DS      1
0001             999 BAUD:    DS      1
0001            1000 KEYOK:   DS      1
0001            1001 ESCP:    DS      1
0001            1002 SHCON:   DS      1
0001            1003 RETLIN:  DS      1
0001            1004 SCNLIN:  DS      1
                1005          END

PUBLIC SYMBOLS

EXTERNAL SYMBOLS

USER SYMBOLS
ADX     A 04CD   ARND   A 046D   BAUD   A 0FEC   BDLK   A 05C5   BTDIS  A 0F80   BYPASS A 008F
CAPLOC  A 022E   CGRT   A 03AD   CHREC  A 024E   CHRPUT A 0477   CLEAR  A 02CF   CLLINE A 0415
CLRLIN  A 0327   CLRST  A 02D5   CLSCR  A 03E4   CNT0   A 6000   CNT1   A 6001   CNT2   A 6002
CNTM    A 6003   CNWD55 A 1803   CONCL  A 02FD   CRTM   A 1000   CRTS   A 1001   CURAD  A 0FE8
CURSX   A 0FE2   CURSY  A 0FE1   DOWN   A 02AE   ESCP   A 0FEE   ESKAP  A 03A5   ESSQ   A 027B
FMFD    A 03CA   FRAME  A 0167   GD18   A 0359   HOME   A 0397   IN75   A 00F9   INT75  A 1401
KEYDWN  A 0FEA   KEYINP A 0121   KEYOK  A 0FED   KEYS   A 0131   KPTK   A 0084   KYCHNG A 01BA
KYLKUP  A 0507   LAST   A 0FD0   LDCUR  A 03B8   LEFT   A 036E   LINNUM A 0019   LINTAB A 04D5
LNFD    A 03F6   LNFD1  A 03FC   LNGTH  A 0050   LOADX  A 03EF   LOC80  A 0FE5   LOOPF  A 00A7
LPKBD   A 0098   NOVER  A 038D   NTOVER A 0364   OK1    A 049C   OK7    A 015C   ONBOT  A 0453
POPDAT  A 0034   PORTA  A 1800   PORTB  A 1801   PORTC  A 1802   RDKB   A 018F   RETLIN A 0FF0
RXRDY   A 0113   SAVKEY A 01AF   SCNLIN A 0FF1   SCR    A 0211   SETUP  A 010F   SHCON  A 0FEF
STBAUD  A 00DC   STKEY  A 0223   STPTR  A 0FE0   TOPAD  A 0FE3   TPDIS  A 0800   TRANS  A 0148
UP1     A 01E9   UPCUR  A 0333   USCHR  A 0FE7   USTD   A A000   USTF   A A001

ASSEMBLY COMPLETE,  NO ERRORS
```

intel

ARTICLE REPRINT

AR-178

September 1983

Low-Cost CRT Control Does More with Less

Thomas Rossi, Applications Manager
Peripheral Components Division

Reprinted with permission from Electronic Design', Vol.29, No.9; copyright Hayden Publishing Co., Inc., 1981

©Intel Corporation, 1981.

ORDER NUMBER: 210507-001

Fewer parts make a microprocessor-based CRT controller cost-effective, and interrupt-driven software cuts overhead on the system's CPU.

Low-cost CRT control does more with less

The multitude of components and the CPU overhead long associated with cathode-ray-tube controllers are rapidly becoming conspicuous by their absence. In particular, an intelligent terminal based on Intel's iAPX 88/10 (8088) microprocessor and 8276 small-system CRT controller eliminates all but 22 of the nearly 40 chips required by other CRT controllers (even those with microprocessors and integrated peripherals). It also cuts overhead on the processor to less than 25%, so that the 88/10 is free to implement such intelligent terminal functions as local data processing.

The iAPX 88/10 implementation supplies characters directly to the 8276 by means of interrupt-driven software, eliminating the need for a direct-memory-access (DMA) controller. The design interfaces directly with standard CRT monitors, contact-closure keyboards, and RS-232C serial-communication links (asynchronous or bisynchronous), to provide a complete stand-alone operator interface.

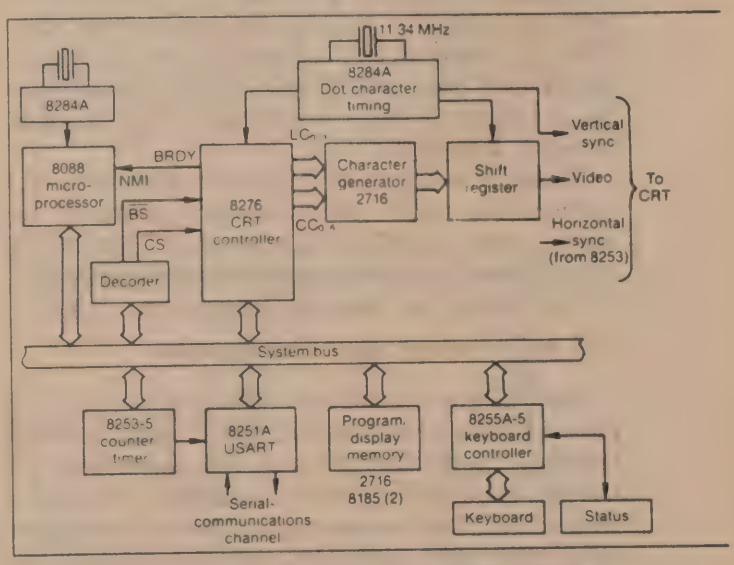
Although the primary design goal—implementing a low-cost CRT terminal—has excluded some useful CRT features, these are easily made available through additional external hardware. For example, composite video is added with two TTL packages, a transistor, and some resistors and capacitors. Another simple option involves the two general-purpose attribute outputs on the 8276 and lets users select any one of four colors on a color monitor.

Basic system configuration and architecture

Central to the 22-chip CRT controller design is an iAPX 88/10 8-bit microprocessor operating at 5 MHz and supported by two 8185 1-kbit × 8 static RAMs and a 2716 control software PROM (Fig. 1). An 8251A programmable communication interface provides synchronous or asynchronous serial communications. Three manual switches on the PC board select the baud rate, and one of the 8253's three independent programmable interval timers generates the 8251A's baud-rate clock under software control.

The three PC-board switches are monitored by the iAPX 88/10 to determine the desired baud rate. When the CPU detects a change in the switch positions, the 8253 is loaded with the appropriate count for the new baud rate.

An 8255A provides three 8-bit parallel I/O ports. Two I/O ports contribute keyboard scanning, and the



1. Intelligent terminals, built with Intel's iAPX 88/10 (8088) microprocessor and new 8276 small-system CRT controller, take this basic configuration to reduce parts count and minimize overhead on the system CPU.

Thomas Rossi, Applications Mgr. Peripheral Components
Intel Corp.
3065 Bowers Ave., Santa Clara, CA 95051

third port senses option-switch settings and the vertical-retrace signal from the 8276 (for CRT synchronization upon reset).

The CRT dot and character timing is generated by an 8284A clock generator. Another 8253 timer provides the appropriate horizontal-retrace timing for the CRT monitor. In its programmable one-shot mode, this timer generates a 32-μs horizontal-retrace pulse for the CRT monitor (Ball Brothers TV-12). A simple user-initiated change in the software will modify this delay time to suit different CRT monitors. The third and last timer in the 8253 is available for any user-defined need.

A 2716 EPROM on the controller board serves as a user-programmable character generator. A shift register transforms the data from the character EPROM into a serial-bit stream to illuminate dots on the CRT screen. The 2716 character generator helps to create special symbols and characters for word processing, industrial-control applications, or foreign-language displays.

The controller hardware is divided into processor and support, serial and parallel I/O, and CRT-control sections. The processor and support section consists of an iAPX 88/10 microprocessor, which is supported by two 8185 1-kbit × 8 static-RAM devices, and another 2716 EPROM (containing 2 kbytes of control firmware). The iAPX 88/10 uses a 15-MHz crystal (with an 8284A) to operate at a 5-MHz clock rate. The 8185 memories attach directly to the iAPX 88/10 multiplexed bus. An 8282 latches eight address lines (A_0-A_7) from the multiplexed bus for 2716-program memory access (Fig. 2).

The serial and parallel I/O section of the terminal includes the 8255A programmable peripheral interface, and the CRT section contains the 8276 CRT controller and support circuits. All of the controller's I/O operations are memory mapped (see table).

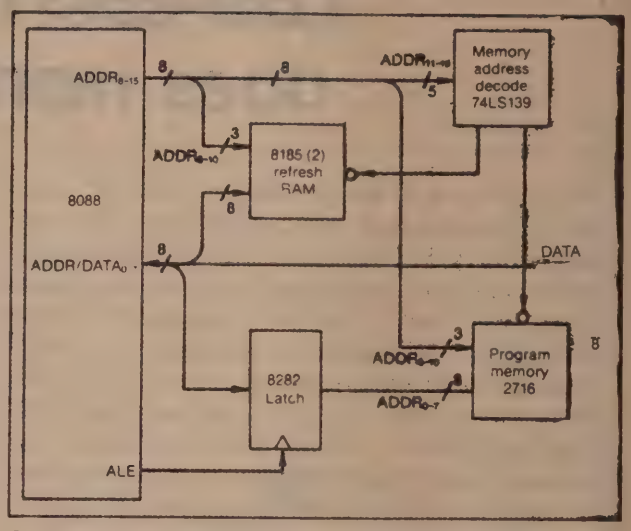
How the controller board communicates

The CRT-controller board communicates to computer systems and other CRT units through a serial interface. Both RS-232C and TTL-compatible interfaces are available at the J_1 connector. The unit's standard software supports eight data-transmission rates: 9600, 4800, 2400, 1200, 600, 300, 150, and 110 baud. These rates are switch-selectable on the PC board. Since the baud-rate clock is generated by an 8253, baud rates may be easily modified in software.

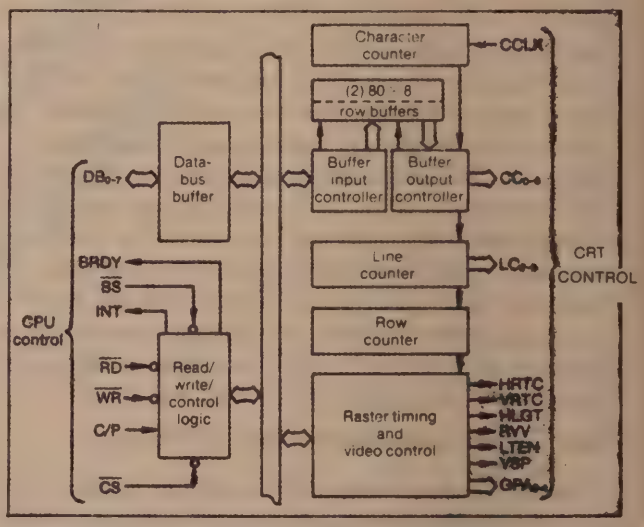
Keyboard scanning is supported through the A and B ports of a 8255A programmable peripheral interface. Therefore, low-cost unencoded keyboards can be used. The eight scan lines (port B) and eight return lines (port A) support a 64-contact closure-key matrix. The three switches attached to port C permit baud-rate selection. Four general-purpose

Memory map of controller I/O operations

Address range	Selected device	Comments
00000 - 00003	RAM	Interrupt vector
00004 - 00029	RAM	Stack, local variables
00030 - 007FF	RAM	Display buffer
01000 - 01001	8276 $\overline{CS}$	8276 command/status
01900	8276 $\overline{BS}$	8276 row buffers
12000 - 12001	8251A	Serial channel
14000 - 14003	8253	Baud-rate timer
18000 - 18003	8255A	Keyboard, switches
FF800 - FFFFF	2716	Program storage



2. The processor and support section of the intelligent terminal's hardware contains two 8185 RAMs attached directly to the iAPX 88/10 multiplexed bus. An 8282 latches eight address lines (A_0-A_7) from the multiplexed bus for 2716-program memory access.



3. Here are the major functional blocks of the 8276 programmable CRT controller. This device permits software specification of most CRT-screen format characteristics (cursor position, characters/row, rows/frame).

inputs on port C permit the software to sense depression of the caps-lock key, the control key, and the shift key, as well as the position of the line/local switch. The last input on port C senses the status of the vertical retrace (VRTC) output of the 8276, so that the controller can synchronize with the CRT display on power up or after a hardware reset.

All keyboard I/O connects to the terminal board by means of a 40-pin header on its edge. All seven option-switch inputs are also brought to the connector, so that option switches may be installed on the keyboard if desired.

Software specifies the screen format

The CRT display is controlled by the 8276 programmable CRT controller (Fig. 3). With this device, most CRT screen-format characteristics—such as the cursor position, the number of characters per row, and the number of rows per frame—can be specified through software. The 8276 handles all display timing including retrace time delays.

In the current design, 2000 characters are displayed on the CRT screen (25 rows of 80 characters). Each character is formed as a 5 × 7-dot matrix within a larger 7 × 10 matrix (Fig. 4). Other screen formats (e.g., 16 rows of 64 characters) can be easily implemented with a few software changes and no hardware changes.

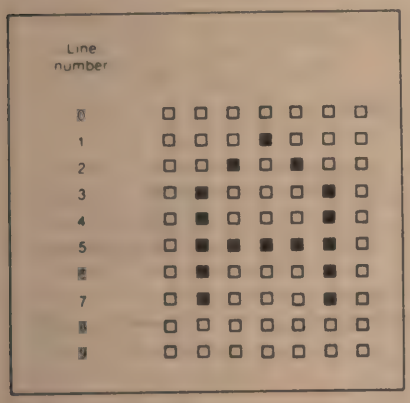
The 8276 contains two 80-character row buffers (see "Row Buffers Reduce System Overhead"). While one buffer displays the current character line on the screen, the 8276 fills the other row buffer from memory. This data transfer begins when the 8276 issues a data request (by means of the BRDY pin), causing an interrupt to the CPU. In response to this interrupt, the CPU activates the RAM's CS and RD inputs, while simultaneously activating the 8276 BS and WR inputs (Fig. 5). Through this technique, a single bus cycle suffices to transfer each byte from the RAM into the CRT row buffer. After the row buffer is filled, the CPU exits the interrupt-service routine.

But the 8276 can do more than simply paint characters on a CRT screen. Its end-of-row-stop buffer-loading code allows the control software to blank individual display lines. Also, the end-of-the-screen-stop buffer-loading code initiates an erase to the end of the screen.

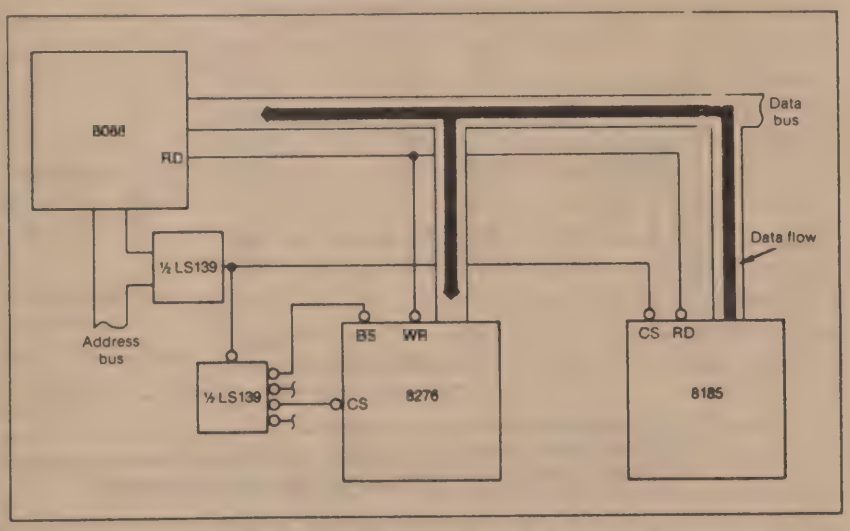
The 8276 supports software selection of visible-field "attributes" that can blink, underline, or highlight (intensify) characters on the screen and can reverse the video-character fields (black letters on a white background). Two general-purpose attribute outputs are provided to control the user-defined display capabilities.

Hardware provides three support functions

The 8276 is supported by three hardware functions: a dot/character-clock oscillator, an EPROM character generator, and a character-shift register (Fig. 6). The dot/character-clock oscillator consists of an 8284A operating at 11.34 MHz and providing an 88.2-ns dot clock. A 74LS163 divides this clock by 7 to generate a 1.62-MHz (617-ns) character clock.



4. The dot-matrix character font used in the low-cost CRT controller creates a 5 × 7 character in a 7 × 10 matrix (example shown is an upper-case A). Top and bottom lines are blanked for character separation, and the remaining line is reserved for cursor/underline display.



5. Row-buffer loading for the 8276 begins when a single 8088 string instruction moves data bytes from the 8185 RAM to the 8276 row buffer. The 8088 CPU "thinks" it is loading the AX register.

The 8276 is programmed to display one raster line every 61.7 μs—a complete character line every 617 μs (ten raster lines). The 8276 is also programmed to refresh the screen every 16.7 ms (60 Hz).

Each character row consists of ten raster lines. Seven lines display the 5 × 7-character matrix, two lines are blanked for row spacing, and one line displays the cursor and underline.

The 8276 uses the line count (LC_0-LC_3) outputs to indicate the current raster line during the display of each character. These outputs, combined with the character-code outputs (CC_0-CC_6), are sent to the 2716, which generates the dot pattern for display. This dot pattern is loaded into the shift register and is serially clocked for display by the 11.34-MHz dot clock.

During the vertical-retrace interval, the row buffer for the first line of the next frame is loaded by the iAPX 88/10. When the frame starts, the 8276 outputs the first character on its CC_0-CC_6 pins; the LC outputs are all zero. Exactly 617 ns later, the next character code is emitted by the 8276. This process continues every 617 ns until all 80 characters have been output. Then the 8276 generates a horizontal-retrace pulse, which is converted to the appropriate pulse width for the CRT monitor by the 8253.

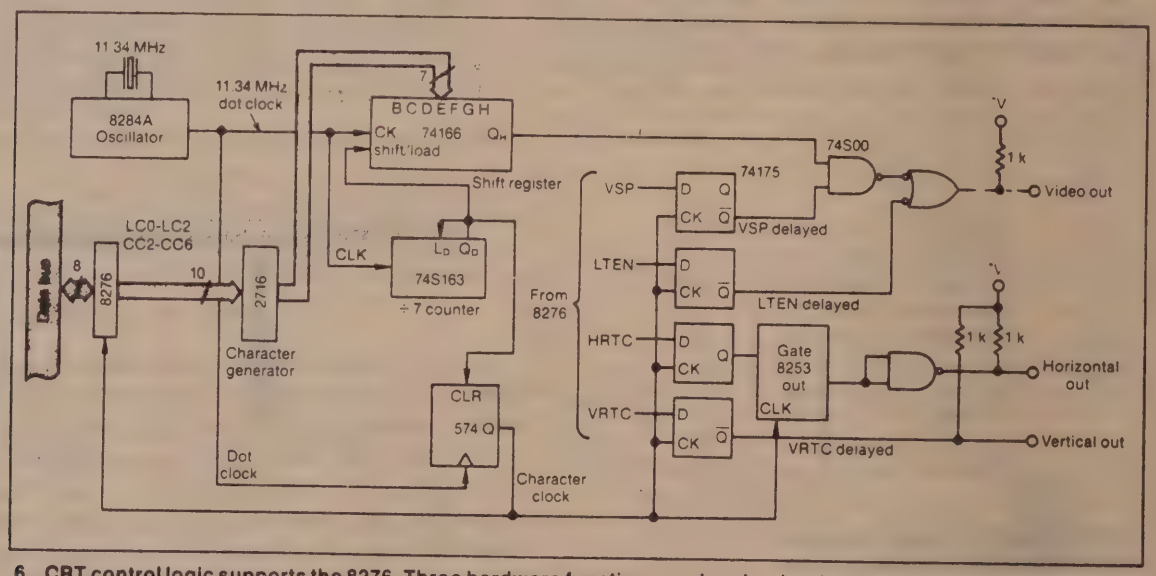
At the end of the first raster line, the 8276 increments the LC outputs. The next nine raster lines are similar to the first—the 8276 outputs the same 80 character codes on the CC_0-CC_6 pins for each of the raster lines, and the LC outputs are incremented after each raster line.

While the ten raster lines are being displayed, the 8276 is also filling the next row buffer. After the tenth raster line is completed, the 8276 resets the LC count and outputs character codes for the second row on the CC_0-CC_6 pins. As this row is displayed, the first row buffer is filled with information for the third row. The 8276 alternates row buffers until all 25 rows are displayed. At this time, the vertical-retrace signal is activated, and the scanning process is repeated for the next frame.

During display, the 8276 automatically activates the video-suppress pin (VSP) and/or light-enable outputs (LTEN), as appropriate, to control retrace blanking, generate the cursor, or underline characters.

Software is split between two priorities

The software for the CRT controller is divided into high and low-priority sections. The high-priority "foreground" software is activated each time the 8276 requests (through the iAPX 88/10 NMI interrupt) that an 80-character row buffer be filled. The 8276 row buffer is filled by performing 80 sequential memory reads. As each read is performed, the



6. CRT control logic supports the 8276. Three hardware functions are involved: a dot/character clock oscillator, an EPROM character generator, and a character-shift register.

Row buffers reduce system overhead

If no row buffer is present, the CRT controller must go to main memory to fetch every character during every dot scan line. Thus, the central processing unit is forced to relinquish the system bus 90 to 95% of the time. That CPU inactivity (overhead) greatly degrades total system performance and efficiency. CRT terminals using this approach are typically limited to between 1200 and 2400 baud on their serial-communications channels.

However, with the 8276's row-buffered architecture, the CRT controller need only access the main memory once for each displayed character row. This approach reduces system bus overhead for CRT refreshing to 25% (maximum). The CPU is then free to perform other local-processing functions, for instance, processing data at 9600 baud on a serial-communications channel.

```
        PUSHF                   ; save registers
        PUSH    SI              ; used by
        PUSH    CX              ; subroutine

        MOV     SI,CURAD        ; point to current line
        ADD     SI, OFFSET      ;
        CLD                     ; auto increment
        MOv     CX,40
    REP LODS    WDPTR           ; move 40 words
        CMP     SI, LAST        ; check for end of screen
        JNZ     KTPK            ; jump if not at end
        MOv     SI,TOPDIS       ; end-set to top
   KTPK MOv     CURAD,SI

        POP     CX              ; restore
        POP     SI              ;
        POPF                    ;
```

7. A screen-refresh routine illustrates how the iAPX 88/10 load-string (LODS) instruction fills an 8276 row buffer. The 15 lines take 167 μs and are run every ten CRT lines (every 617 μs).

```
XOR     AX, AX          ; clear AX
MOV     BX, ESCTBL      ; load table, pointer
MOV     AL, USCHR       ; read character
CMP     AL, 41H         ; check for 41H
JL      SETUP           ; not valid
CMP     AL, 48H         ; check for 48H
JG      SETUP           ; not valid
XLAT                    ; translate to routine
                          address
JMP     (AX)
```

8. This routine checks the keyboard character to see if it is a valid escape-sequence command (41H through 48H). If the character is valid, a translate table jumps to a service routine. With the powerful iAPX 88/10 translate instruction, the service routine takes just 7 μs.

hardware automatically sends a write (over buffer-select and write pins) to the 8276.

The simultaneous memory-read and 8276-write commands transfer characters from the 8185 RAM to the 8276 in a single memory cycle—without a direct-memory-access (DMA) controller. The 80 reads are under the control of the CPU load string (LODS) instruction, which handles 40 word loads with iAPX 88/10 code (Fig. 7). The complete refresh sequence for one line requires approximately 167 μs. As a result, processor overhead for refresh operations is approximately 27%.

Foreground software also involves keyboard scanning that is performed only at the end of each display frame (after 25 rows or 16.7 ms). If a key depression is noted during one of these scans, the information is stored for further background processing. An iAPX 88/10 routine checks the character to determine whether it is a valid escape-character command (Fig. 8). In this procedure, the iAPX 88/10's translate instruction (XLAT) takes care of table lookup.

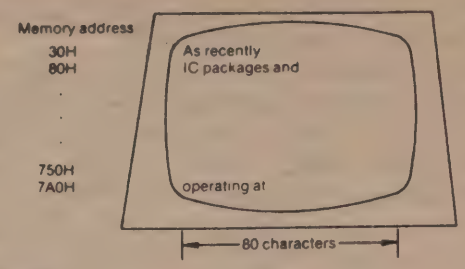
The low-priority software section handles "background" processing. It monitors the 8251A serial I/O port and provides processing for characters entered via the keyboard or with the serial interface. Background software executes continuously except when interrupted for the higher-priority foreground processing.

Cumbersome scrolling technique avoided

A refresh-buffer memory stores all 2000 characters that can be displayed on the CRT screen. The foreground software transfers one row (of 80 characters) at a time to the 8276. Two pointers are employed during normal operation. Under the control of foreground processing software, the current-row pointer contains the address of the next row to be displayed. This pointer must always be correct, so that a row can be transferred to the 8276 when requested. The buffer pointer contains the address of the next CRT buffer location to be written into (from either the keyboard or the serial port). Controlled by the background software, the buffer pointer indicates the cursor's actual location.

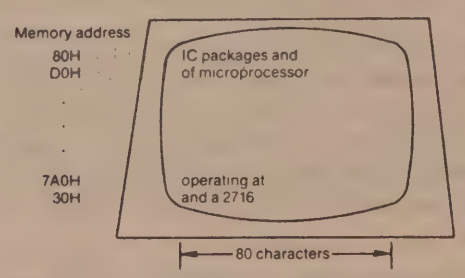
The simplest refresh-buffer organization associates the first memory address with the upper left position on the CRT screen. All other characters are stored sequentially (Fig. 9). But this method makes CRT screen scrolling difficult. Scrolling requires that each display line be moved up one row. The top line of the CRT is lost, the bottom line is blanked, and the cursor is placed at the beginning of the bottom line.

With this fixed sequential organization, all characters in the refresh buffer must be moved forward



Memory address	ASCII	Hex	Notes
30H	A	41	First character, first row
31H	s	73	
32H	space	20	
.	.	.	
.	.	.	
80H	I	49	First character, second row
81H	C	43	
.	.	.	
.	.	.	
7FFH	space	20	Last screen character

9. This memory/screen-character relationship exists when all characters are stored sequentially, making scrolling difficult.



Memory address	ASCII	Hex	Notes
30H	a	61	First character, 25th row
31H	n	6E	
32H	d	64	
.	.	.	
.	.	.	
80H	I	49	First character, first row
81H	C	43	
.	.	.	
.	.	.	
7FFH	space	20	Last character, first row

10. If sequential memory orientation is retained but characters do not have to be moved in memory, scrolling can be much more efficient. Here, scrolling is accomplished simply by changing the display-start pointer. The memory/screen-character relationship is shown after a scroll of one line from the positions illustrated in Fig. 9.

by 80 characters (memory locations) to scroll the screen. (Each line moves up one row on the CRT and the last 80 characters in the buffer are blanked.) Moving 1920 characters each time the screen scrolls a single line is very slow and cumbersome.

The low-cost CRT controller avoids this problem with a slight modification of the fixed-sequential scrolling technique. Here, sequential memory orientation is retained while the need to move characters in memory is eliminated. This approach requires an additional display-start pointer that points to the memory location of the first character to be displayed.

At system initialization, the display-start pointer is set to 30H, the buffer-start address. During each vertical-retrace interval, the current-row pointer is initialized from the display-start pointer. Scrolling is performed by merely changing the display-start pointer.

For a single row scroll, the display-start pointer moves ahead 80 characters to location 80H, and the first 80 characters in the buffer are blanked. During the next vertical retrace, the foreground software sets the current-row pointer to the display-start location (80H), and begins transferring characters to the 8276 from this address.

The character in memory-location 80H (previously the first character in the second row) now occupies the first display position on the CRT screen (first character of the first row). When the foreground software reaches the end of the display buffer, the next row is read from the beginning of the buffer (location 30H). Thus, the first 80 characters in the buffer appear on the last display row (Fig. 10).

Each subsequent scroll moves the display start pointer forward by 80 characters. Buffer operations automatically "roll over" to the physical beginning of the buffer after passing the last buffer location.

Since the row-by-row character display is controlled by iAPX 88/10 software, other display techniques may be used. In particular, a linked list structure is extremely adaptable to word-processing and text-editing functions. This method allows each row within a file to be changed independently of other rows.

Because the rows are linked or "chained together" by pointers, rows may be easily inserted or deleted by simply changing pointers. To display a CRT frame, the processor simply follows the pointer chain from one row to the next.□

How useful?	Circle
Immediate design application	547
Within the next year	548
Not applicable	549

intel

PRELIMINARY

82720
GRAPHICS DISPLAY CONTROLLER

- Displays Low-to-High Resolution Images
- Draws Characters, Points, Lines, Arcs, and Rectangles
- Supports Monochrome, Gray Scale, or Color Displays
- Zooms, Pans and Windows Through a 4 Mpixel Display Memory
- Extremely Flexible Programmable Screen Display, Blanking, and Sync Formats
- Compatible with Intel's Microprocessor Families
- High-Level Commands Off Load Host Processor from Bit Map Loading and Screen Refresh Tasks
- Supports Graphics, Character, and Mixed Display Modes

FUNCTIONAL DESCRIPTION

Introduction

The 82720 Graphics Display Controller (GDC) is an intelligent microprocessor peripheral designed to drive high-performance raster-scan computer graphics and character CRT displays. Positioned between the video display memory and Intel microprocessor bus, the GDC performs the tasks needed to generate the raster display and manage the display memory. Processor software overhead is minimized by the GDC's sophisticated instruction set, graphics figure drawing, and DMA transfer capabilities. The display memory directly supported by the GDC can be configured in any number of formats and sizes up to 256K 16-bit words. The display can be zoomed and partitioned screen areas can be independently scrolled and panned. With its light pen input and multiple controller capability, the GDC is ideal for most computer graphics applications. Systems implemented with the GDC can be designed to be compatible with standards such as VDI, NAPLPS, GKS, Core, or custom implementations.

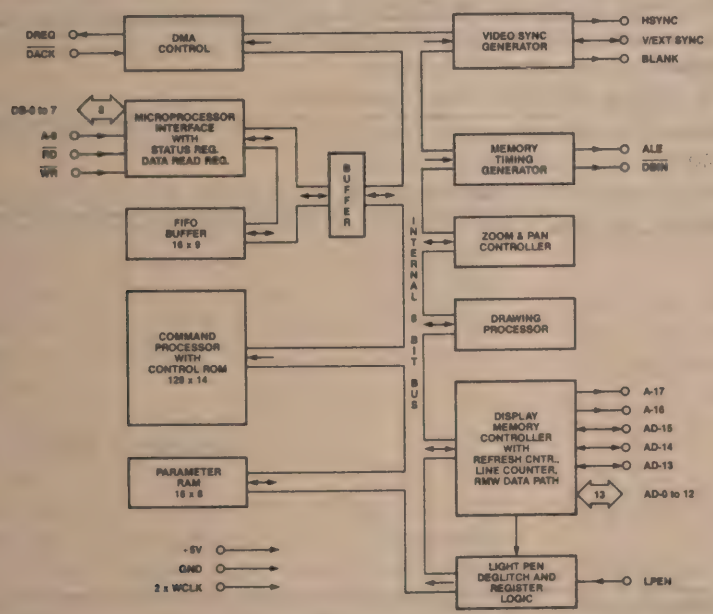


Figure 1. Block Diagram

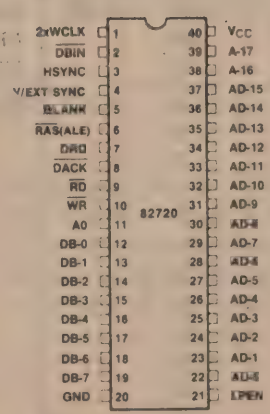


Figure 2. Pin Configuration

Intel Corporation Assumed No Responsibility for the Use of Any Circuitry Other Than Circuitry Embodied in an Intel Product. No Other Circuit Patent Licenses are Implied. Information contained herein supersedes previously published specifications on these devices from Intel.
©Intel Corporation, 1985

January 1985
Order Number: 210655-003

Table 1. Pin Description

Symbol	Pin No.	Type	Name and Description
2XWCLK	1	I	**Clock Input**
$\overline{DBIN}$	2	O	**Display Bus Input:** Read strobe output used to read display memory data into the GDC.
HSYNC	3	O	**Horizontal Sync:** Output used to initiate the horizontal retrace of the CRT display.
V/EXT SYNC	4	I/O	**Vertical Sync:** Output used to initiate the vertical retrace of the CRT display. In slave mode, this pin is an input used to synchronize the GDC with the master raster timing device.
BLANK	5	O	**Blank:** Output used to suppress the video signal.
$\overline{RAS}$ (ALE)	6	O	**Row Address Strobe (Address Latch Enable):** Output used to start the control timing chain when used with dynamic RAMs. When used with static RAMs, this signal is used to demultiplex the display address/data bus.
DRQ	7	O	**DMA Request:** Output used to request a DMA transfer from a DMA controller (8237) or I/O processor (8089).
$\overline{DACK}$	8	I	**DMA Acknowledge:** Input used to acknowledge a DMA transfer from a DMA controller or I/O processor.
$\overline{RD}$	9	I	**Read:** Input used to strobe GDC Data into the microprocessor.
$\overline{WR}$	10	I	**Write:** Input used to strobe microprocessor data into the GDC.
A0	11	I	**Register Address:** Input used to select between commands and data read or written.
DB0 DB1 DB2 DB3 DB4 DB5 DB6 DB7	12 13 14 15 16 17 18 19	I/O	**Bidirectional Microprocessor Data Bus Line:** Input enabled by $\overline{WR}$. Output enabled by $\overline{RD}$.
GND	20		**Ground.**
V_{CC}	40		**+5V Power Supply**
A_{17}	39	O	**Graphics Mode:** Display Address Bit 17 Output **Character Mode:** Cursor and Line Counter Bit 4 Output **Mixed Mode:** Cursor and Image Mode Flag
A_{16}	38	O	**Graphics Mode:** Display Address Bit 16 Output **Character Mode:** Line Counter Bit 3 Output **Mixed Mode:** Attribute Blink and Line Counter Reset
AD_{15} AD_{14} AD_{13}	37 36 35	I/O	**Graphics Mode:** Display Address/Data Bits 13–15 **Character Mode:** Line Counter Bits 0–2 Output **Mixed Mode:** Display Address/Data Bits 13–15
AD_{12} AD_{11} AD_{10} AD_9 AD_8 AD_7 AD_6 AD_5 AD_4 AD_3 AD_2 AD_1 AD_0	34 33 32 31 30 29 28 27 26 25 24 23 22	I/O	**Display Address/Data Bits 0–12**
LPEN	21	I	**Light Pen Detect Input**

FUNCTIONAL DESCRIPTION (Continued)

Microprocessor Bus Interface

Control of the GDC by the system microprocessor is achieved through an 8-bit bidirectional interface. The status register is readable at any time. Access to the FIFO buffer is coordinated through flags in the status register.

Command Processor

The contents of the FIFO are interpreted by the command processor. The command bytes are decoded, and the succeeding parameters are distributed to their proper destinations within the GDC. The bus interface has priority over the command processor when both access the FIFO simultaneously.

DMA Control

The DMA Control circuitry in the GDC coordinates data transfers when using an external DMA controller. The DMA Request and Acknowledge handshake lines interface with an 8257 or 8237 DMA controller or 8089 I/O processor, so that display data can be moved between the microprocessor memory and the display memory.

Parameter RAM

The 16-byte RAM stores parameters that are used repetitively during the display and drawing processes. In character mode, the RAM holds the partitioned display area parameters. In graphics mode, the RAM also holds the drawing pattern and graphics character.

Video Sync Generator

Based on the clock input, the sync logic generates the raster timing signals for almost any interlaced, non-interlaced, or "repeat field" interlaced video format. The generator is programmed during the idle period following a reset. In video sync slave mode, it coordinates timing between the GDC and another video source.

Memory Timing Generator

The memory timing circuitry provides two memory cycle types: a two-clock period refresh cycle and the read-modify-write (RMW) cycle which takes four clock periods. The memory control signals needed to drive the display memory devices are easily generated from the GDC's $\overline{RAS}$(ALE) and $\overline{DBIN}$ outputs.

Zoom and Pan Controller

Based on the programmable zoom display factor and the display area parameters in the parameter RAM, the zoom and pan controller determines when to advance to the next memory address for display refresh and when to go on to the next display area. A horizontal zoom is produced by slowing down the display refresh rate while maintaining the video sync rates. Vertical zoom is accomplished by repeatedly accessing each line a number of times equal to the horizontal repeat. Once the line count for a display area is exhausted, the controller accesses the starting address and line count of the next display area from the parameter RAM. The system microprocessor, by modifying a display area starting address, allows panning in any direction, independent of the other display areas.

Drawing Processor

The drawing processor contains the logic necessary to calculate the addresses and positions of the pixels of the various graphics figures. Given a starting point and the appropriate drawing parameters, the drawing processor needs no further assistance to complete the figure drawing.

Display Memory Controller

The display memory controller's tasks are numerous. Its primary purpose is to multiplex the address and data information in and out of the display memory. It also contains the 16-bit logic units used to modify the display memory contents during RMW cycles, the character mode line counter, and the refresh counter for dynamic RAMs. The memory controller apportions the video field time between the various types of cycles.

Light Pen Debouncer

Only if two rising edges on the light pen input occur at the same point during successive video fields are the pulses accepted as a valid light pen detection. A status bit indicates to the system microprocessor that the light pen register contains a valid address.

System Operation

The GDC is designed to work with Intel microprocessors to implement high-performance computer graphics systems. System efficiency is maximized through partitioning and a pipelined architecture. At the lowest level, the GDC generates the basic video

raster timing, including sync and blanking signals. Partitioned areas on the screen and zooming are also accomplished at this level. At the next level, video display memory is modified during the figure drawing operations and data moves. Third, display memory address are calculated pixel by pixel as drawing progresses. Outside the GDC at the next level, preliminary calculations are done to prepare drawing parameters. At the fifth level, the picture must be represented as a list of graphics figures drawable by the GDC. Finally, this representation must be manipulated, stored and communicated. The GDC takes care of the high-speed and repetitive tasks required to implement graphics systems.

GENERAL OVERVIEW

In order to minimize system bus loading, the 82720 uses a private video memory for storage of the video image. Up to 512K bytes of video memory can be directly supported. For example, this is sufficient capacity to store a 2048 × 2048 pixel × 1 bit image. Images can be generated on the screen by:

—Drawing Commands
—Program-Controlled Transfers
—DMA Transfers from System Memory

The 82720 can be configured to support a wide variety of graphics applications. It can support:

—High Dot Rates
—Color Planes
—Horizontal Split Screen
—Character-oriented Displays
—Multiplexed Graphic and Character Display

GRAPHIC DISPLAY CONFIGURATIONS

The 82720 provides the flexibility to handle a wide variety of graphic applications. This flexibility results from having its own private video memory for storage of the graphics image. The organization of this memory determines the performance, the number of bits/pixel and the size of the display. Several different video memory organizations are examined in the following paragraphs.

In the simplest 82720 system, the memory can store up to a 2048 × 2048 × 1 bit image. It can display a 1024 × 1024 × 1 bit section of the image at a maximum dot rate of 44 MHz, or 88 MHz in wide mode. In this configuration, only 1 bit/pixel is used.

By partitioning the memory into multiple banks, color, gray scale and higher bandwidth displays can be supported. By adding various amounts of external logic, many cost/performance tradeoffs for both display and drawing are realizable.

The video memory can be partitioned into 4 banks, each 1024 x 1024 bits. By selecting all 4 memory banks during display, 4 bits/pixel can be provided by a single 82720. Each bank of video memory contributes 1 bit to each pixel. This configuration can support color monitors, again with a maximum dot shift rate of 44 or 88 MHz.

Higher performance may be achieved by using multiple 82720s. Multiple 82720s can be used to support mutliple display windows, increased drawing speed, or increased bits per pixel. For display windows, each 82720 controls one window of the display. For increased drawing speed, multiple 82720s are operated in parallel. For increased bits/pixel, each 82720 contributes a portion of the number of bits necessary for a pixel.

CHARACTER DISPLAY CONFIGURATION

Although the 82720 is intended primarily for raster-scan graphics, it can be used as a character display controller. The 82720 can support up to 8K by 13 bits of private video memory in this configuration (1 character = 13 bits). This is sufficient memory to store 4 screens of data containing 25 rows by 80 characters. The 82720 can display up to 256 characters per row. Smooth vertical scrolling of each of 4 independent display partitions is also supported.

MIXED DISPLAY CONFIGURATION

The GDC can support a mixed display system for both graphic and character information. This capability allows the display screen to be partitioned between graphic and character data. It is possible to switch between one graphic display window and one character display window with raster line resolution. A maximum of 256K bytes of video memory is supported in this mode: half is for graphic data, half is for character data. In graphic mode, a one megapixel image can be stored and displayed. In character mode, 64K, 16-bit characters can be stored.

DETAILED OPERATIONAL DESCRIPTION

The GDC can be used in one of three basic modes —Graphics Mode, Character Mode and Mixed Mode. This section of the data sheet describes the following for each mode:

1. Memory organization
2. Display timing
3. Special Display functions
4. Drawing and writing

Graphics Mode Memory Organization

The Display Memory is organized into 16-bit words (32-bit words in wide mode). Since the display memory can be larger than the CRT display itself, two width parameters must be specified: display memory width and display width. The Display width (in words) is selected by a parameter of the Reset command. The Display memory width (in words) is selected by a parameter of the Pitch command. The height of the Display memory can be larger than the display itself. The height of the Display is selected by a parameter of the Reset command. The GDC can directly address up to 4Mbits (0.5Mbytes) of display RAM in graphics mode.

Graphics Mode Display Timing

All raster blanking and display timings of the GDC are a function of the input clock frequency. Sixteen or 32 bits of data are read from the RAM and loaded into a shift register in each two clock period display cycle. The Address and Data busses of the GDC are multiplexed. In the first part of the cycle, the address of the word to be read is latched into an external demultiplexer. In the second part of the cycle the data is read from the RAM and loaded into the shift register. Since all 16 (32) bits of data are to be displayed, the dot clock is 8 × (16 ×) the GDC clock or 16 × (32 ×) the Read cycle rate.

Parameters of the Reset or Sync command determine the horizontal and vertical front porch, sync pulse, and back porch timings. Horizontal parameters are specified as multiples of the display cycle time, and vertical parameters as a multiple of the line time.

Another Reset command parameter selects interlaced or non-interlaced mode. A bit in the parameter RAM can define Wide Display Mode. In this mode, while data is being sent to the screen, the display address counter is incremented by two rather than one. This allows the display memory to be configured to deliver 32 bits from each display read cycle.

The V Sync command specifies whether the V Sync Pin is an input or an output. If the V Sync Pin is an output, the GDC generates the raster timing for the display and other CRT controllers can be synchronized to it. If the V Sync pin is an input, the GDC can be synchronized to any external vertical Sync signal.

Graphics Mode Special Display Functions:

WINDOWING

The GDC's Graphics Mode Display can be divided into two windows on the screen, upper and lower. The windows are defined by parameters written into the GDC's parameter RAM. Each window is specified by a starting address and a window length in lines. If the second window is not used, the first window parameters should be specified to be the same as the active display length.

ZOOMING

A parameter of the GDC's zoom command allows zooming by effectively increasing the size of the dots on the screen. This is accomplished vertically by repeating the same display line. The number of times it is repeated is determined by the display zoom factor parameter. Horizontally, zoom is accomplished by extending each display word cycle and displaying fewer words per line, according to the zoom factor. It is the responsibility of the microprocessor controlling the GDC to provide the shift register clock circuitry with the zoom factor required to slow down the shift registers to the appropriate speed. The frequency of the 2XWCLK should not be changed. The zoom factor must be set to a known state upon initialization.

PANNING

Panning is accomplished by changing the starting address of the display window. In this way, panning is possible in any direction, vertically on a line by line basis and horizontally on a word by word basis.

Graphics Mode Drawing and Writing

The GDC can draw solid or patterned lines, arcs, circles, rectangles, slanted rectangles, characters, slanted characters, filled rectangles. Direct access to the bit map is also provided via the DMA Commands and the Read or Write data commands.

MEMORY MODIFICATION

All drawing and writing functions take place at the location in the display RAM specified by the cursor. The cursor is not displayed in Graphics Mode. The cursor location is modified by the execution of drawing, reading or writing commands. The cursor will move to the bit following the last bit accessed.

Each bit is drawn by executing a Read-Modify-Write cycle on the display RAM. These R/M/W cycles normally require four 2XWCLK cycles to execute. If the display zoom factor is greater than two, each R/M/W cycle will be extended to the width of a display cycle. Write Data (WDAT), Read Data (RDAT), DMA write (DMAW) and DMA read (DMAR) commands can be used to examine or modify one to 16 bits in each word during each R/M/W cycle. All other graphics drawing commands modify one bit per R/M/W cycle.

An internal 16-bit Mask register determines which bit(s) in the accessed word are to be modified. A one in the Mask register allows the corresponding bit in the display RAM to be modified by the R/M/W cycle. A zero in the Mask register prevents the GDC from modifying the corresponding bit in the display RAM.

The mask must be set by the Mask Command prior to issuing the WDAT or DMAW command. The Mask register is automatically set by the CURS command and manipulated by the graphics commands.

The display RAM bits can be modified in one of four ways. They can be set to 1, reset to 0, complemented or replaced by a pattern.

When replace by a pattern mode is selected, lines, arcs and rectangles will be drawn using the 16-bit pattern in parameter RAM bytes 8 and 9.

In set, reset, or complement mode, parameter RAM bytes 8 and 9 act as another level of masking for line arc and rectangle drawing. As each 16-bit segment of the line or arc is drawn, it is checked against the pattern in the parameter RAM. If the pattern RAM bit is a one, the display RAM bit will be set, reset, or complemented per the proper modes. If the pattern RAM bit is a zero, the display RAM bit won't be modified.

When replace by pattern mode is selected, the graphics character and fill commands will cause the 8 x 8 pattern in parameter RAM bytes 8 to 15 to be written directly into the display RAM in the appropriate locations.

In set, reset, or complement mode, the 8 x 8 pattern in parameter RAM bytes 8 to 15 act as a mask pattern for graphics character or fill commands. If the appropriate parameter RAM bit is set, the display RAM bit will be modified. If the parameter RAM bit is zero, the display RAM bit will not be modified. These modes are selected by issuing a WDAT command without parameters before issuing graphics commands. The pattern in the parameter RAM has no effect on WDAT, RDAT, DMAW, or DMAR operations.

READING AND DRAWING COMMANDS

After the modification mode has been set and the parameter RAM has been loaded, the final drawing parameters are loaded via the figure specify (FIGS) command. The first parameter specifies the direction in which drawing will occur and the figure type to be drawn. This parameter is followed by one to five more parameters depending on the type of character to be drawn.

The direction parameter specifies one of eight octants in which the drawing or reading will occur. The effect of drawing direction on the various figure types is shown in Figure 9.

RDAT, WDAT, DMAR, and DMAW Operations move through the Display memory as shown in the "DMA" Column.

The other parameters required to set up figure reading or drawing are shown in Figure 3.

DRAWING TYPE	DC	D	D2	D1	DM
INITIAL VALUE*	0	8	8	−1	−1
LINE	$\lvert\Delta I\rvert$	$2\lvert\Delta D\rvert - \lvert\Delta I\rvert$	$2(\lvert\Delta D\rvert - \lvert\Delta I\rvert)$	$2\lvert\Delta D\rvert$	–
ARC**	$r\sin\phi\rfloor$	$r-1$	$2(r-1)$	−1	$r\sin\theta\rfloor$
RECTANGLE	3	A−1	B−1	−1	A−1
AREA FILL	B−1	A	A	–	–
GRAPHIC CHARACTER***	B−1	A	A	–	–
WRITE DATA	W−1	–	–	–	–
DMAW	D−1	C−1	–	–	–
DMAR	D−1	C−2	(C−2)/2†	–	–
READ DATA	W	–	–	–	–

*INITIAL VALUES FOR THE VARIOUS PARAMETERS ARE LOADED WHEN THE FIGS COMMAND BYTE IS PROCESSED.
**CIRCLES ARE DRAWN WITH 8 ARCS, EACH OF WHICH SPAN 45°, SO THAT SIN ϕ = 1/√2 AND SIN θ = 0.
***GRAPHIC CHARACTERS ARE A SPECIAL CASE OF BIT-MAP AREA FILLING IN WHICH B AND A ≤8. IF A = 8 THERE IS NO NEED TO LOAD D AND D2.

WHERE:
−1 = ALL ONES VALUE.
ALL NUMBERS ARE SHOWN IN BASE 10 FOR CONVENIENCE. THE GDC ACCEPTS BASE 2 NUMBERS (2s COMPLEMENT NOTATION WHERE APPROPRIATE).
– = NO PARAMETER BYTES SENT TO GDC FOR THIS PARAMETER.
ΔI = THE LARGER OF Δx OR Δy.
ΔD = THE SMALLER OF Δx OR Δy.
r = RADIUS OF CURVATURE, IN PIXELS.
ϕ = ANGLE FROM MAJOR AXIS TO END OF THE ARC. $\phi \leq 45°$.
θ = ANGLE FROM MAJOR AXIS TO START OF THE ARC. $\theta \leq 45°$.
⌉ = ROUND UP TO THE NEXT HIGHER INTEGER.
⌋ = ROUND DOWN TO THE NEXT LOWER INTEGER.
A = NUMBER OF PIXELS IN THE INITIALLY SPECIFIED DIRECTION.
B = NUMBER OF PIXELS IN THE DIRECTION AT RIGHT ANGLES TO THE INITIALLY SPECIFIED DIRECTION.
W = NUMBER OF WORDS TO BE ACCESSED.
C = NUMBER OF BYTES TO BE TRANSFERRED IN THE INITIALLY SPECIFIED DIRECTION. (TWO BYTES PER WORD IF WORD TRANSFER MODE IS SELECTED.)
D = NUMBER OF WORDS TO BE ACCESSED IN THE DIRECTION AT RIGHT ANGLES TO THE INITIALLY SPECIFIED DIRECTION.
DC = DRAWING COUNT PARAMETER WHICH IS ONE LESS THAN THE NUMBER OF RMW CYCLES TO BE EXECUTED.
DM = DOTS MASKED FROM DRAWING DURING ARC DRAWING.
† = NEEDED ONLY FOR WORD READS.

Figure 3. Drawing Parameter Details

After the parameters have been set, line, arc, circle, rectangle or slanted rectangle drawing operations are initiated by the Figure Draw (FIGD) command. Character, slanted character, area fill and slanted area fill drawing operations are initiated by the Graphics Character Draw (GCHRD) command. DMA transfers are initiated by the DMA Read or Write (DMAR or DMAW) commands. Data Read Operations are initiated by the Read Data (RDAT) Command. Data Write Operations are initiated by writing a parameter after the WDAT command.

The area fill operation steps and repeats the 8 × 8 graphics character pattern draw operation to fill a rectangular area. If the size of the rectangle is not an integral number of 8 x 8 pixels, the GDC will automatically truncate the pattern at the edges furthest from the starting point.

The Graphics Character Drawing capability can be modified by the Graphics Character Write Zoom Factor (GCHR) parameter of the zoom command. The zoom write factor may be set from 1 to 16 (by using from 0 to 15 in the parameter). Each dot will be repeated in memory horizontally and vertically (adjusted for drawing direction) the number of times specified by the zoom factor.

The WDAT command can be used to rapidly fill large areas in memory with the same value. The mask is set to all 1's, and the least significant bit of the WDAT parameter replaces all bits of each word written.

Character Mode Memory Organization

In character mode, the Display memory is organized into up to 8K characters of up to 13 bits each. Wide mode is also available for characters of up to 26 bits.

The display memory can be larger than the display itself. The display width (in characters) is a parameter of the reset command. The display memory width (in characters) is a parameter of the Pitch Command. The height of the display (in lines) is a parameter of the Reset Command. The display memory height is determined by dividing the number of display memory words by the pitch.

In character mode, the display works almost exactly as it does in graphics mode. The differences lie in the fact that data read from the display RAM is used to drive a character generator as well as attribute logic if desired. In Character mode, address bits 13–16 become line counter outputs used to select the proper line of the character generator, and the address 17 output becomes the cursor and line counter MSB output.

Character Mode Display Timing

In character mode, the display timing works as it does in graphics mode. In addition, the Address 17 output becomes cursor output. The characteristics of the cursor are defined by parameters of the cursor and Character Characteristics (CCHAR) command. One bit allows the cursor output to be enabled or disabled. The height of the cursor is programmable by selecting the top and bottom line between which the cursor will appear. The blink rate is also programmable. The parameter selects the number of frame times that the cursor will be inactive and active, resulting in a 50% duty cycle cursor blinking at 2× the period specified by the parameter.

The cursor output pin also provides the line counter bit 4 signal, which is valid 10 clocks after the trailing edge of HSYNC.

Character Mode Special Display Functions

WINDOWING

The GDC's Character Mode display can be partitioned into one to four windows on the screen. The windows are defined by parameters written into the GDC's Parameter RAM. Each window is specified by a starting address and a window length in lines.

If windowing is not required, the first window length should be specified to be the same as the active display length.

ZOOMING AND PANNING

In character mode, zooming and pan handling commands function the same way as in Graphics Mode.

Character Mode Drawing and Writing

The GDC can read or write characters of up to 13 bits into or out of the Display RAM.

All reading and writing functions take place at the display RAM location specified by the cursor. The cursor location can be read by issuing the CURD command. The cursor can be moved anywhere within the display memory by the CURS command. The cursor location is also modified by the execution of character read or write commands.

Each character is written or read via a Read/Modify/Write cycle. The mask register contents determine which bit(s) in the character are modified. The mask register can be used to change character codes without modifying attribute bits or vice-versa. The Replace with pattern, Set, Reset and Complement

modes work exactly as they do in graphics mode, with the exception that the parameter RAM Pattern is not used. The pattern used is a parameter of the WDAT command.

The Figure Specify (FIGS) command must be set to Character Display mode, as well as specify the direction the cursor will be moved by read or write data commands.

In character mode, the FIGD and GCHRD commands are not used.

Mixed Mode Memory Organization

In mixed mode, the display memory is organized into two banks of up to 64K words of 16 bits each (32 bits in wide mode).

The display height and width are programmable by the same Reset or Sync command parameters as in the graphics and character modes. The display memory width (in words) is a parameter of the Pitch Command and the height of the display memory is determined by dividing the number of display memory words by the pitch.

An image mode signal is used to switch the external circuitry between graphics and character modes in two display windows.

In a graphics window, the GDC works as it does in pure graphics mode, but on a smaller total memory space (64K words vs 512K words).

In a character window, the GDC works as it does in pure character mode, but the line counter must be implemented externally. The counter is clocked by the horizontal sync pulse and reset by a signal supplied by the GDC.

In mixed mode, the GDC provides both a cursor and an attribute blink timing signal.

Mixed Mode Display Timing

In mixed mode, each word in a graphic area is accessed twice in succession. The AW parameter of the Reset or Sync command should be set to twice its normal value, and the video shift register load signal must be suppressed during the extra access cycle.

In addition, A16 becomes a Multiplexed Attribute and Clear Line Counter signal and A17 becomes a multiplexed cursor and image mode signal. A16 provides an active high line counter reset signal which is valid 10 clocks after the trailing edge of HSYNC. During the active display line time, A16 provides blink timing for external attribute circuitry. This signal blinks at 1/2 the blink rate of the cursor with a 75% on, 25% off duty cycle. A17 provides a signal which selects between graphics or character display, which is also valid 10 clocks after the trailing edge of HSYNC. During the active display time, A17 provides the cursor signal. The cursor timing and characteristics are defined in exactly the same way as in pure character mode.

Mixed Mode Special Display Functions

WINDOWING

The GDC supports two display windows in mixed mode. They can independently be programmed into either graphics or character mode determined by the state of two bits in the parameter RAM. The window location in display memory and size are also determined by parameters in the parameter RAM.

ZOOMING AND PANNING

In mixed mode, zooming and panning commands function the same as in graphics and character mode.

Mixed Mode Drawing and Writing

In mixed mode, the GDC can write or draw in exactly the same ways as in both graphics and character modes. In addition, the FIGS command has a parameter GD (Graphics Drawing Flag) which sets the image mode signal to select the proper RAM bank.

DEVICE PROGRAMMING

The GDC occupies two addresses on the system microprocessor bus through which the GDC's status register and FIFO are accessed. Commands and parameters are written into the GDC FIFO and are differentiated by address bit A0. The status register or the FIFO can be read as selected by the address line.

A0	READ	WRITE
0	STATUS REGISTER	PARAMETER INTO FIFO
1	FIFO READ	COMMAND INTO FIFO

Figure 4. GDC Microprocessor Bus Interface Registers

Commands to the GDC take the form of a command byte followed by a series of parameter bytes as needed for specifying the details of the command. The command processor decodes the commands, unpacks the parameters, loads them into the appropriate registers within the GDC and initiates the required operations.

The commands available in the GDC can be organized into five categories as described in figure 5.

VIDEO CONTROL COMMANDS
1. RESET: RESETS THE GDC TO ITS IDLE STATE.
2. SYNC: SPECIFIES THE VIDEO DISPLAY FORMAT.
3. VSYNC: SELECTS MASTER OR SLAVE VIDEO SYNCHRONIZATION MODE
4. CCHAR: SPECIFIES THE CURSOR AND CHARACTER ROW HEIGHTS.

DISPLAY CONTROL COMMANDS
1. START: ENDS IDLE MODE AND UNBLANKS THE DISPLAY.
2. BCTRL: CONTROLS THE BLANKING AND UNBLANKING OF THE DISPLAY.
3. ZOOM: SPECIFIES ZOOM FACTORS FOR THE DISPLAY AND GRAPHICS CHARACTERS WRITING.
4. CURS: SETS THE POSITION OF THE CURSOR IN DISPLAY MEMORY.
5. PRAM: DEFINES STARTING ADDRESSES AND LENGTHS OF THE DISPLAY AREAS AND SPECIFIES THE EIGHT BYTES FOR THE GRAPHICS CHARACTER.
6. PITCH: SPECIFIES THE WIDTH OF THE X DIMENSION OF DISPLAY MEMORY.

DRAWING CONTROL COMMANDS
1. WDAT: WRITES DATA WORDS OR BYTES INTO DISPLAY MEMORY.
2. MASK: SETS THE MASK REGISTER CONTENTS.
3. FIGS: SPECIFIES THE PARAMETERS FOR THE DRAWING PROCESSOR.
4. FIGD: DRAWS THE FIGURE AS SPECIFIED ABOVE.
5. GCHRD: DRAWS THE GRAPHICS CHARACTER INTO DISPLAY

MEMORY. DATA READ COMMANDS
1. RDAT: READS DATA WORDS OR BYTES FROM DISPLAY MEMORY.
2. CURD: READS THE CURSOR POSITION.
3. LPRD: READS THE LIGHT PEN ADDRESS.

DMA CONTROL COMMANDS
1. DMAR: REQUESTS A DMA READ TRANSFER.
2. DMAW: REQUESTS A DMA WRITE TRANSFER.

Figure 5. GDC Command Summary

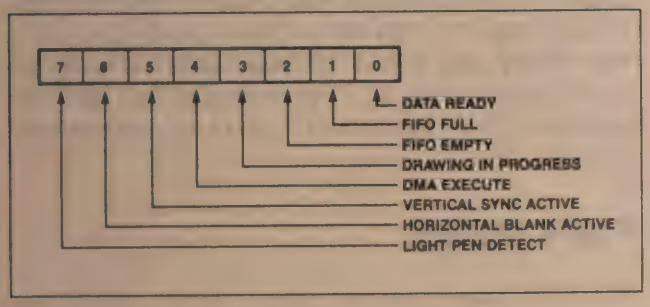


Figure 6. Status Register (SR)

Status Register Flags

SR-7: Light Pen Detect: When this bit is set to 1, the light pen address (LAD) register contains a deglitched value that the system microprocessor may read. This flag is reset after the 3-byte LAD is moved into the FIFO in response to the light pen read command.

SR-6: Horizontal Blanking Active: A 1 value for this flag signifies that horizontal retrace blanking is currently underway.

SR-5: Vertical Sync: Vertical retrace sync occurs while this flag is a 1. The vertical sync flag coordinates display format modifying commands to the blanked interval surrounding vertical sync. This eliminates display disturbances.

SR-4: DMA Execute: This bit is a 1 during DMA data transfers.

SR-3: Drawing in Progress: While the GDC is drawing a graphics figure, this status bit is a 1.

SR-2: FIFO Empty: This bit and the FIFO Full flag coordinate system microprocessor accesses with the GDC FIFO. When it is 1, the Empty flag ensures that all the commands and parameters previously sent to the GDC have been processed.

SR-1: FIFO Full: A 1 at this flag indicates a full FIFO in the GDC. A 0 ensures that there is room for at least one byte. This flag needs to be checked before each write into the GDC.

SR-0: Data Ready: When this flag is a 1, it indicates that a byte is available to be read by the system microprocessor. This bit must be tested before each read operation. It drops to a 0 while the data is transferred from the FIFO into the microprocessor interface data register.

FIFO Operation & Command Protocol

The first-in, first-out buffer (FIFO) in the GDC handles the command dialogue with the system microprocessor. This flow of information uses a half-duplex technique, in which the single 16-location FIFO is used for both directions of data movement, one direction at a time. The FIFO's direction is controlled by the system microprocessor through the GDC's command set. The microprocessor coordinates these transfers by checking the appropriate status register bits.

The command protocol used by the GDC requires the differentiation of the first byte of a command sequence from the succeeding bytes. This first byte contains the operation code and the remaining bytes carry parameters. Writing into the GDC causes the FIFO to store a flag value alongside the data byte to signify whether the byte was written into the command or the parameter address. The command processor in the GDC tests this bit as it interprets the entries in the FIFO.

The receipt of a command byte by the command processor marks the end of any previous operation. The number of parameter bytes supplied with a command is cut short by the receipt of the next command byte. A read operation from the GDC to the microprocessor can be terminated at any time by the next command.

The FIFO changes direction under the control of the system microprocessor. Commands written into the GDC always put the FIFO into write mode if it wasn't in it already. If it was in read mode, any read data in the FIFO at the time of the turnaround is lost. Commands which require a GDC response, such as RDAT, CURD and LPRD, put the FIFO into read mode after the command is interpreted by the GDC's command processor. Any commands and parameters behind the read-evoking command are discarded when the FIFO direction is reversed.

Read-Modify-Write Cycle

Data transfers between the GDC and the display memory are accomplished using a read-modify-write (RMW) memory cycle. The four clock period timing of the RMW cycle is used to: 1) output the address, 2) read data from the memory, 3) modify the data, and 4) write the modified data back into the initially selected memory address. This type of memory cycle is used for all interactions with display memory including DMA transfers, except for the two clock period display and RAM refresh cycles.

The operations performed during the modify portion of the RMW cycle merit additional explanation. The circuitry in the GDC uses three main elements: the Pattern register, the Mask register, and the 16-bit Logic unit. The Pattern register holds the data pattern to be moved into memory. It is loaded by the WDAT command or, during drawing, from the parameter RAM. The Mask register contents determine which bits of the read data will be modified. Based on the contents of these registers, the Logic unit performs the selected operations of REPLACE, COMPLEMENT, SET, or CLEAR on the data read from display memory.

The Pattern register contents are ANDed with the Mask register contents to enable the actual modification of the memory read data, on a bit-by-bit basis. For graphics drawing, one bit at a time from the Pattern register is combined with the Mask. When ANDed with the bit set to a 1 in the Mask register, the proper single pixel is modified by the Logic Unit. For the next pixel in the figure, the next bit in the Pattern register is selected and the Mask register bit is moved to identify the pixel's location within the word. The Execution word address pointer register, EAD, is also adjusted as required to address the word containing the next pixel.

In character mode, all of the bits in the Pattern register are used in parallel to form the respective bits of the modify data word. Since the bits of the character code word are used in parallel, unlike the one-bit-at-a-time graphics drawing process, this facility allows any or all of the bits in a memory word to be modified in one RMW memory cycle. The Mask register must be loaded with 1s in the positions where modification is to be permitted.

The Mask register can be loaded in either of two ways. In graphics mode, the CURS command contains a four-bit dAD field to specify the dot address. The command processor converts this parameter into the one-of-16 format used in the Mask register for figure drawing. A full 16 bits can be loaded into the Mask register using the MASK command. In addition to the character mode use mentioned above, the 16-bit MASK load is convenient in graphics mode when all of the pixels of a word are to be set to the same value.

The Logic unit combines the data read from display memory, the Pattern register, and the Mask register to generate the data to be written back into display memory. Any one of four operations can be selected: REPLACE, COMPLEMENT, CLEAR or SET. In each case, if the respective Mask bit is 0, that particular bit of the read data is returned to memory unmodified. If the Mask bit is 1, the modification is enabled. With the REPLACE operation, the modify data simply takes the place of the read data for modification enabled bits. For the other three operations, a 0 in the modify data allows the read data bit to be returned to memory. A 1 value causes the specified operation to be performed in the bit positions with set Mask bits.

Figure Drawing

The GDC draws graphics figures at the rate of one pixel per read-modify-write (RMW) display memory cycle. These cycles take four clock periods to complete. At a clock frequency of 5 MHz, this is equal to 800 ns. During the RMW cycle the GDC simultaneously calculates the address and position of the next pixel to be drawn.

The graphics figure drawing process depends on the display memory addressing structure. Groups of 16 horizontally adjacent pixels form the 16-bit words

which are handled by the GDC. Display memory is organized as a linearly addressed space of these words. Addressing of individual pixels is handled by the GDC's internal RMW logic.

During the drawing process, the GDC finds the next pixel of the figure which is one of the eight nearest neighbors of the last pixel drawn. The GDC assigns each of these eight directions a number from 0 to 7, starting with straight down and proceeding counterclockwise.

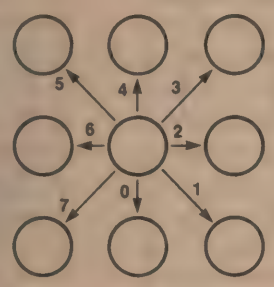


Figure 7. Drawing Directions

Figure drawing requires the proper manipulation of the address and the pixel bit position according to the drawing direction to determine the next pixel of the figure. To move to the word above or below the current one, it is necessary to subtract or add the number of words per line in display memory. This parameter is called the pitch. To move to the word to either side, the Execute word address cursor, EAD, must be incremented or decremented as the dot address pointer bit reaches the LSB or the MSB of the Mask register. To move to a pixel within the same word, it is necessary to rotate the dot address pointer register to the right or left.

Figure 8 summarizes these operations for each direction.

Whole word drawing is useful for filling areas in memory with a single value. By setting the Mask register to all 1s with the MASK command, both the LSB and MSB of the dAD will always be 1, so that the EAD value will be incremented or decremented for each cycle regardless of direction. One RMW cycle will be able to affect all 16 bits of the word for any drawing type. One bit in the Pattern register is used per RMW cycle to write all the bits of the word to the same value. The next Pattern bit is used for the word, etc.

DIR	ADDRESS OPERATION(S)
0	EAD = EAD + P
1	EAD = EAD + P If dAD.MSB = 1 then EAD = EAD + 1 dAD → RR (dAD)
else	dAD = LR(dAD)
2	If dAD.MSB = 1 then EAD = EAD + 1 dAD → RR (dAD)
else	dAD = LR(dAD)
3	EAD = EAD − P If dAD.MSB = 1 then EAD = EAD + 1 dAD → RR (dAD)
else	dAD = LR(dAD)
4	EAD = EAD − P
5	EAD = EAD − P If dAD.LSB = 1 then EAD = EAD − 1 dAD → LR (dAD)
else	dAD = RR(dAD)
6	If dAD.LSB = 1 then EAD = EAD − 1 dAD → LR (dAD)
else	dAD = RR(dAD)
7	EAD = EAD + P If dAD.LSB = 1 then EAD = EAD − 1 dAD → LR (dAD)
else	dAD = RR(dAD)

WHERE
P = PITCH, LR = LEFT ROTATE, RR = RIGHT ROTATE
CAD = CURSOR ADDRESS
dAD = DOT ADDRESS
LSB = LEAST SIGNIFICANT BIT
MSB = MOST SIGNIFICANT BIT

Figure 8. Address Calculation Details

For the various figures, the effect of the initial direction upon the resulting drawing is shown in figure 9.

Note that during line drawing, the angle of the line may be anywhere within the shaded octant defined by the DIR value. Arc drawing starts in the direction initially specified by the DIR value and veers into an arc as drawing proceeds. An arc may be up to 45 degrees in length. DMA transfers are done on word boundaries only, and follow the arrows indicated in the table to find successive word addresses. The slanted paths for DMA transfers indicate the GDC changing both the X and Y components of the word address when moving to the next word. It does not follow a 45 degree diagonal path by pixels.

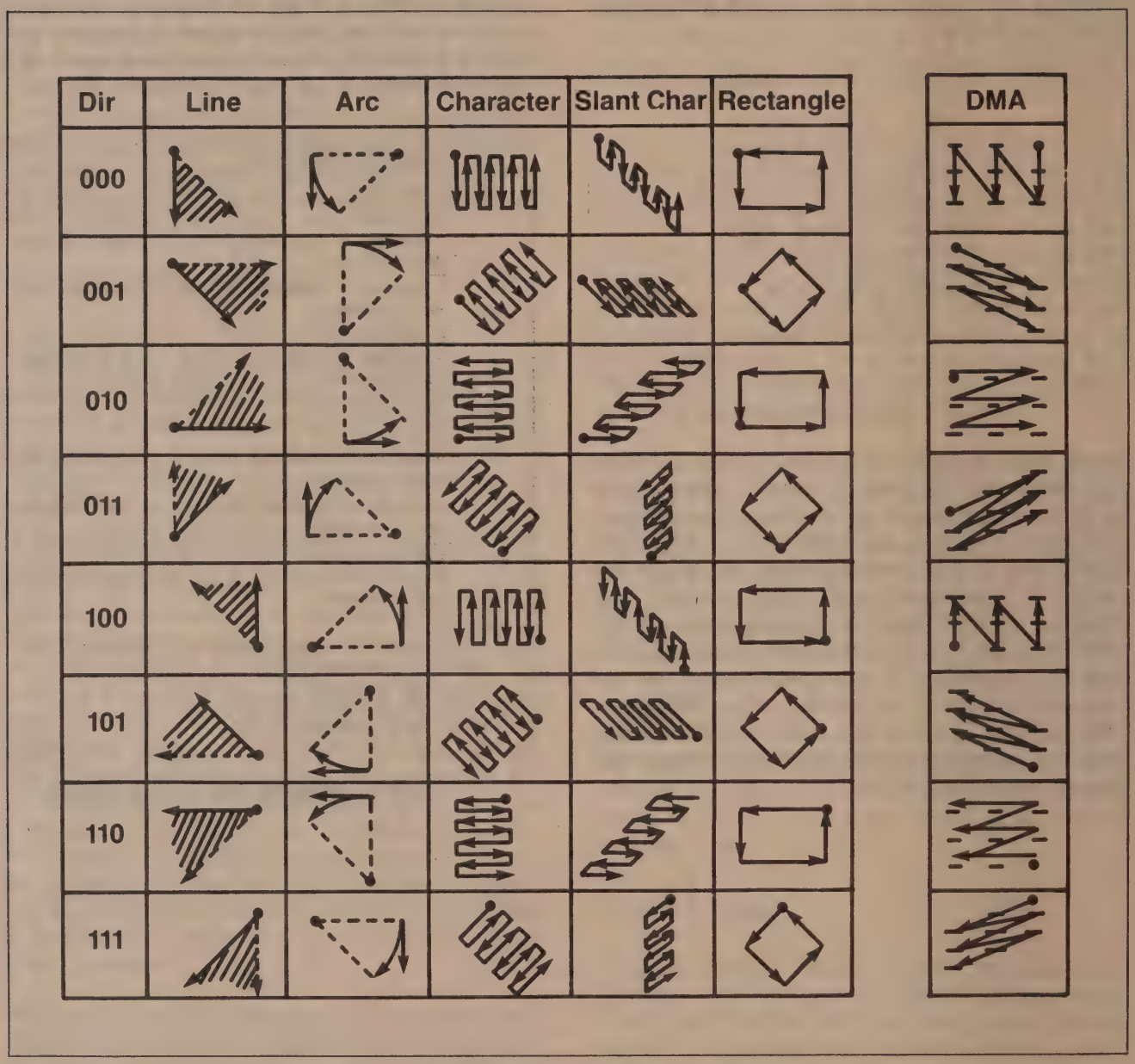


Figure 9. Effect of the Direction Parameter

Drawing Parameters

In preparation for graphics figure drawing, the GDC's Drawing Processor needs the figure type, direction and drawing parameters, the starting pixel address, and the pattern from the microprocessor. Once these are in place within the GDC, the Figure Draw command, FIGD, initiates the drawing operation. From that point on, the system microprocessor is not involved in the drawing process. The GDC Drawing Processor coordinates the RMW circuitry and address registers to draw the specified figure pixel by pixel.

The algorithms used by the processor for figure drawing are designed to optimize its drawing speed. To this end, the specific details about the figure to be drawn are reduced by the microprocessor to a form conducive to high-speed address calculations within the GDC. In this way the repetitive, pixel-by-pixel calculations can be done quickly, thereby minimizing the overall figure drawing time. Figure 3 summarizes the parameters.

Graphics Character Drawing

Graphics characters can be drawn into display memory pixel-by-pixel. The up to 8-by-8 character is loaded into the GDC's parameter RAM by the system microprocessor. Consequently, there are no limitations on the character set used. By varying the drawing parameters and drawing direction, numerous drawing options are available. In area fill applications, a character can be written into display memory as many times as desired without reloading the parameter RAM.

Once the parameter RAM has been loaded with up to eight graphics character bytes by the appropriate PRAM command, the GCHRD command can be used to draw the bytes into display memory starting at the cursor. The zoom magnification factor for writing, set by the zoom command, controls the size of the character written into the display memory in integer multiples of 1 through 16. The bit values in the PRAM are repeated horizontally and vertically the number of times specified by the zoom factor.

The movement of these PRAM bytes to the display memory is controlled by the parameters of the FIGS command. Based on the specified height and width of the area to be drawn, the parameter RAM is scanned to fill the required area.

For an 8-by-8 graphics character, the first pixel drawn uses the LSB of RA-15, the second pixel uses bit 1 of RA-15, and so on, until the MSB of RA-15 is reached. The GDC jumps to the corresponding bit in RA-14 to continue the drawing. The progression then advances toward the LSB of RA-14. This snaking sequence is continued for the other 6 PRAM bytes. This progression matches the sequence of display memory addresses calculated by the drawing processor as shown in figure 9. If the area is narrower than 8 pixels wide, the snaking will advance to the next PRAM byte before the MSB is reached. If the area is less than 8 lines high, fewer bytes in the parameter RAM will be scanned. If the area is larger than 8 by 8, the GDC will repeat the contents of the parameter RAM in two dimensions.

Parameter RAM Contents

The parameters stored in the parameter RAM, PRAM, are available for the GDC to refer to repeatedly during figure drawing and raster-scanning. In each mode of operation the values in the PRAM are interpreted by the GDC in a predetermined fashion. The host microprocessor must load the appropriate parameters into the proper PRAM locations. PRAM loading command allows the host to write into any location of the PRAM and transfer as many bytes as desired. In this way any stored parameter byte or bytes may be changed without influencing the other bytes.

The PRAM stores two types of information. For specifying the details of the display area partitions, blocks of four bytes are used. The four parameters stored in each block include the starting address in display memory of each display area, and its length. In addition, there are two mode bits for each area which specify whether the area is a bit-mapped graphics area or a coded character area, and whether a normal or wide display cycle is to be used for that area.

The other use for the PRAM contents is to supply the pattern for figure drawing when in a bit-mapped graphics area or mode. In these situations, PRAM bytes 8 through 16 are reserved for this patterning information. For line, arc, and rectangle drawing (linear figures) locations 8 and 9 are loaded into the Pattern register to allow the GDC to draw dotted, dashed, etc. lines. For area filling and graphics bit-mapped character drawing locations 8 through 15 are referenced for the pattern or character to be drawn.

Details of the bit assignments are shown on the following pages for the various modes of operation.

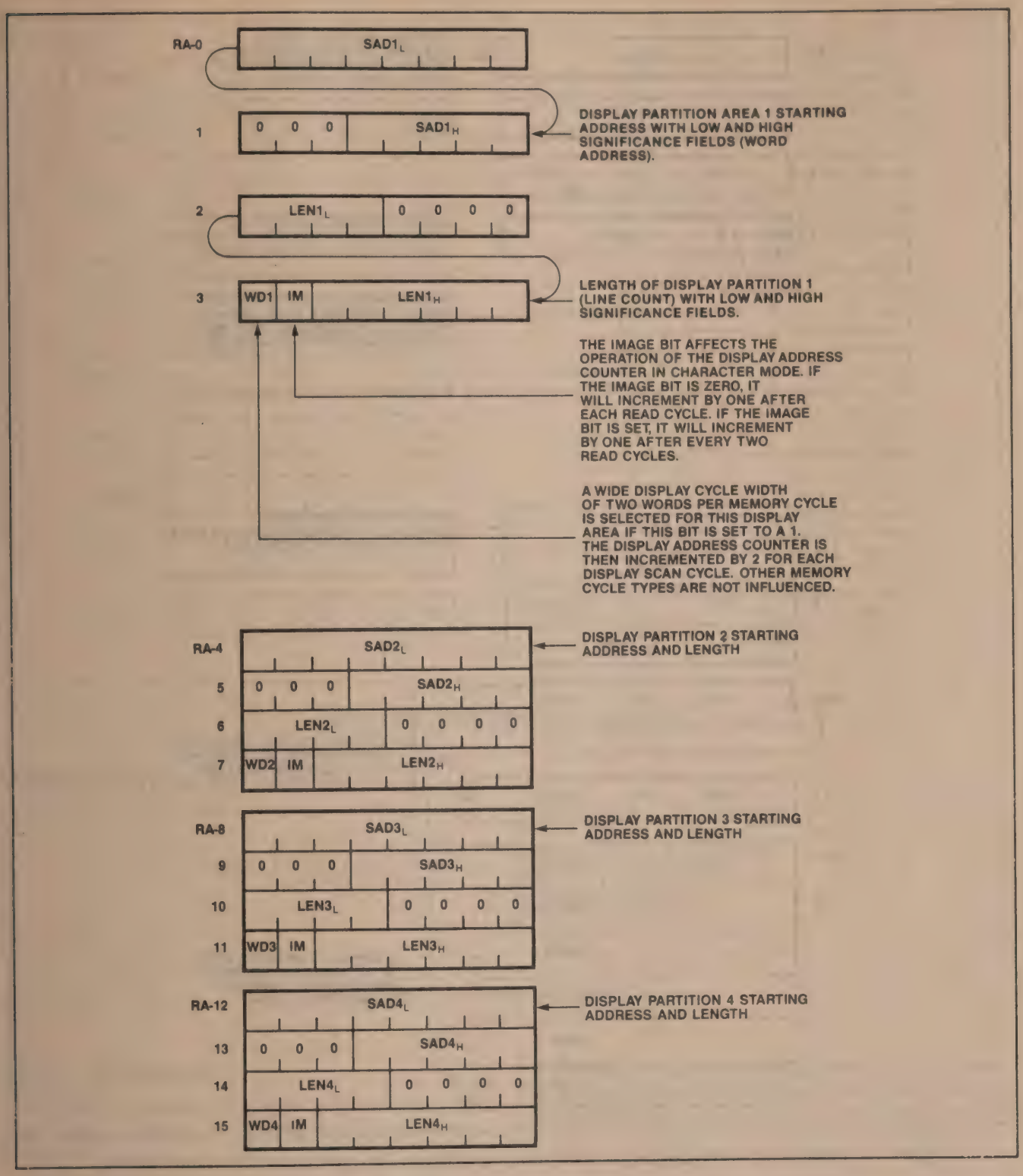


Figure 10. Parameter RAM Contents—Character Mode

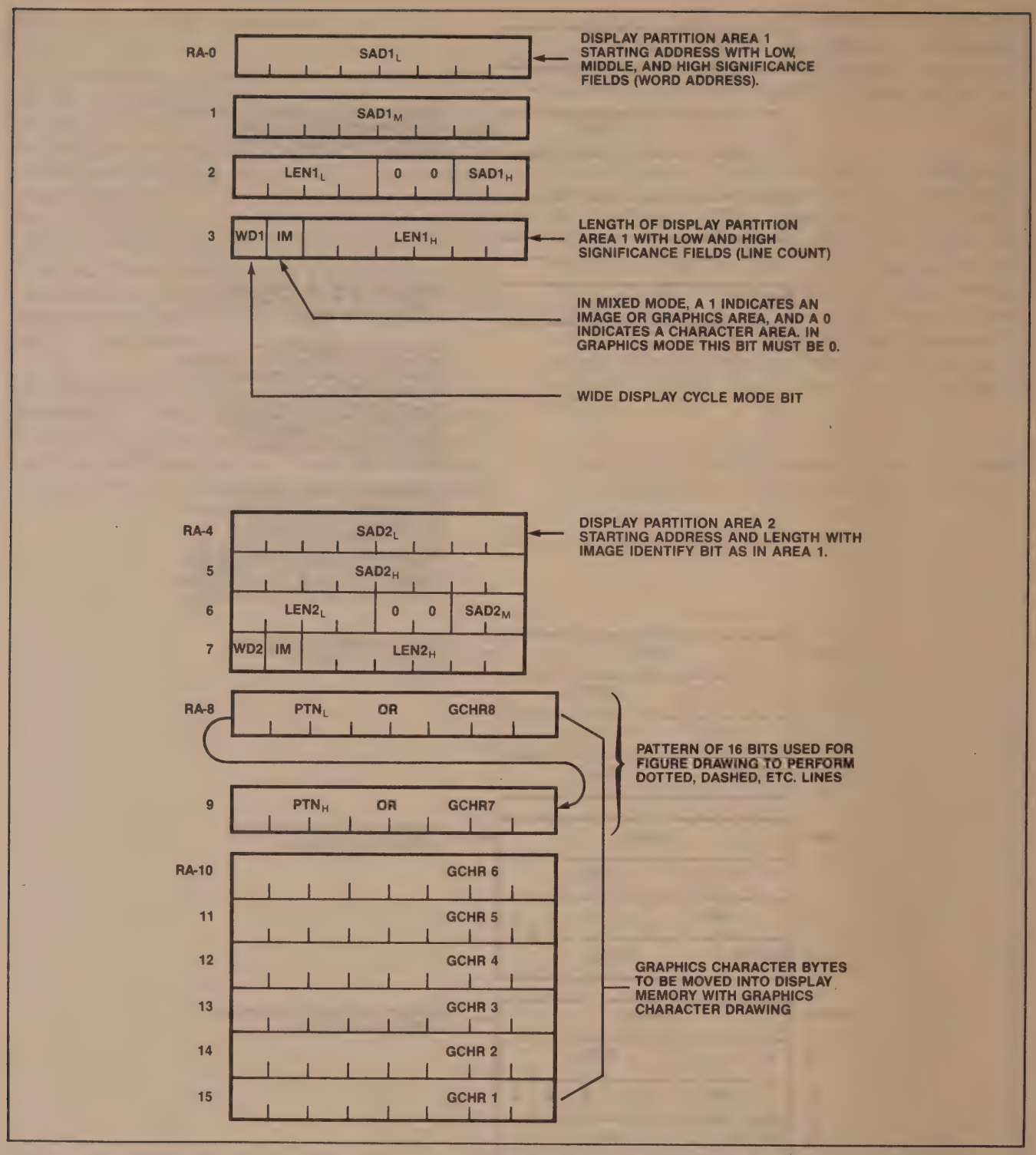


Figure 11. Parameter RAM Contents—Graphics and Mixed Graphics and Character Modes

RESET:	0	0	0	0	0	0	0	0	WDAT:	0	0	1	TYPE	0	MOD	
SYNC:	0	0	0	0	1	1	1	DE	MASK:	0	1	0	0	1	0	1	0
VSYNC:	0	1	1	0	1	1	1	M	FIGS:	0	1	0	0	1	1	0	0
CCHAR:	0	1	0	0	1	0	1	1	FIGD:	0	1	1	0	1	1	0	0
START:	0	1	1	0	1	0	1	1	GCHRD:	0	1	1	0	1	0	0	0
BCTRL:	0	0	0	0	1	1	0	DE	RDAT:	1	0	1	TYPE	0	MOD	
ZOOM:	0	1	0	0	0	1	1	0	CURD:	1	1	1	0	0	0	0	0
CURS:	0	1	0	0	1	0	0	1	LPRD:	1	1	0	0	0	0	0	0
PRAM:	0	1	1	1	SA				DMAR:	1	0	1	TYPE	1	MOD	
PITCH:	0	1	0	0	0	1	1	1	DMAW:	0	0	1	TYPE	1	MOD	

Figure 12. Command Bytes Summary

VIDEO CONTROL COMMANDS

RESET:	0	0	0	0	0	0	0	0	BLANK THE DISPLAY, ENTER IDLE MODE, AND INITIALIZE WITHIN THE GDC: —FIFO —COMMAND PROCESSOR —INTERNAL COUNTERS

Figure 13. Reset Command

RESET COMMAND

This command can be executed at any time and does not modify any of the parameters already loaded into the GDC.

If followed by parameter bytes, this command also sets the sync generator parameters as described below. Idle mode is exited with the START command.

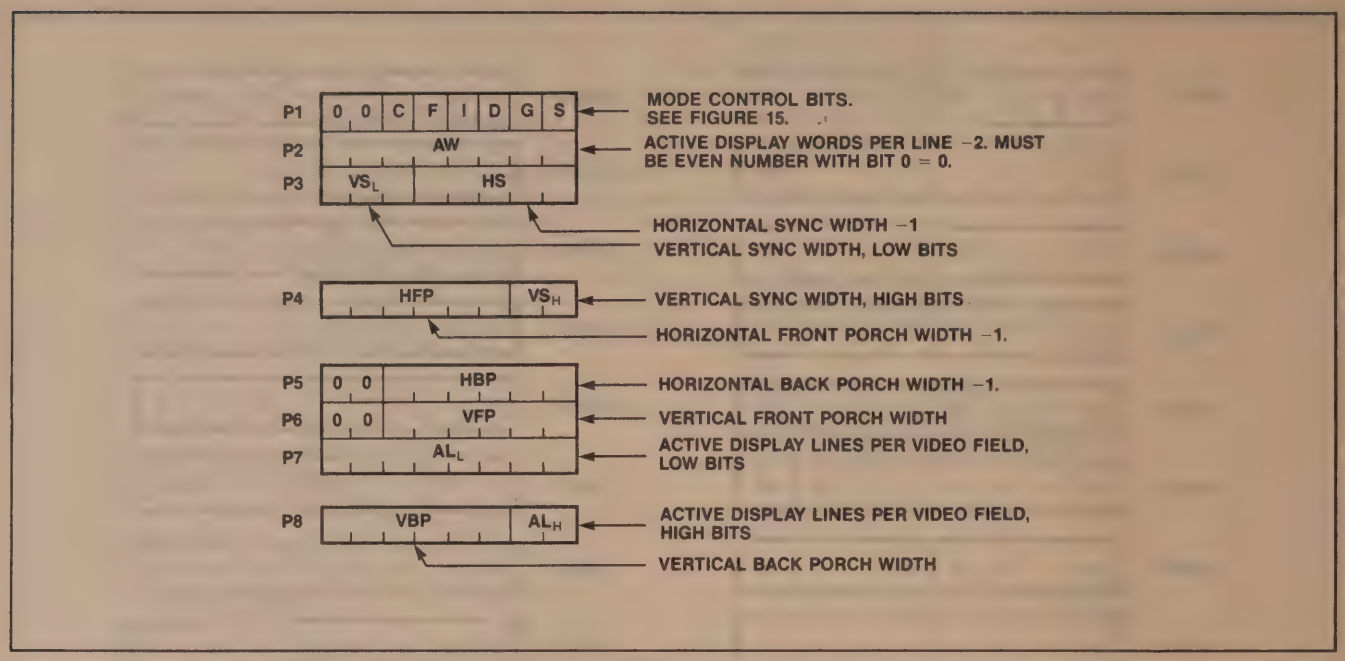


Figure 14. Optional Reset Parameters

In graphics mode, a word is a group of 16 pixels. In character mode, a word is one character code and its attributes, if any.

The number of active words per line must be an even number from 2 to 256.

An all-zero parameter value selects a count equal to 2^n where n = number of bits in the parameter field for vertical parameters.

All horizontal widths are counted in display words. All vertical intervals are counted in lines.

Sync Parameter Constraints

HORIZONTAL FRONT PORCH CONSTRAINTS

1. In general:
   HFP ≥2 words
2. If DMA is used, or the display zoom factor is greater than one in interlaced display mode:
   HFP ≥3 words
3. If the GDC is used in slave mode:
   HFP ≥4 words
4. If the light pen input is used:
   HFP ≥6 words

HORIZONTAL Sync CONSTRAINTS

1. If dynamic RAM refresh is used:
   HS ≥2 words
2. If interlaced display mode is used:
   HS ≥5 words

HORIZONTAL BACK PORCH CONSTRAINTS

1. In general:
   HBP ≥3 words
2. If interlaced display mode is used, or the IMAGE or WIDE mode bits change within one video field:
   HBP ≥ 5 words

MODE CONTROL BITS (FIGURE 15)

Repeat Field Framing:	2 Field Sequence with ½ line offset between otherwise identical fields.
Interlaced Framing:	2 Field Sequence with ½ line offset. Each field displays alternate lines.
Noninterlaced Framing:	1 field brings all of the information to the screen.

Total scanned lines in interlace mode is odd. The sum of VFP + VS + VBP + AL should equal one less than the desired odd number of lines.

Dynamic RAM refresh is important when high display zoom factors or DMA are used in such a way that not all of the rows in the RAMs are regularly accessed during display raster generation and for otherwise inactive display memory.

Access to display memory can be limited to retrace blanking intervals only, so that no disruptions of the image are seen on the screen.

C G	DISPLAY MODE
0 0	MIXED GRAPHICS & CHARACTER
0 1	GRAPHICS MODE
1 0	CHARACTER MODE
1 1	INVALID

I S	VIDEO FRAMING
0 0	NONINTERLACED
0 1	INVALID
1 0	INTERLACED REPEAT FIELD FOR CHARACTER DISPLAYS
1 1	INTERLACED

D	DYNAMIC RAM REFRESH CYCLES ENABLE
0	NO REFRESH—STATIC RAM
1	REFRESH—DYNAMIC RAM

F	DRAWING TIME WINDOW
0	DRAWING DURING ACTIVE DISPLAY TIME AND RETRACE BLANKING
1	DRAWING ONLY DURING RETRACE BLANKING

Figure 15. Mode Control Bits

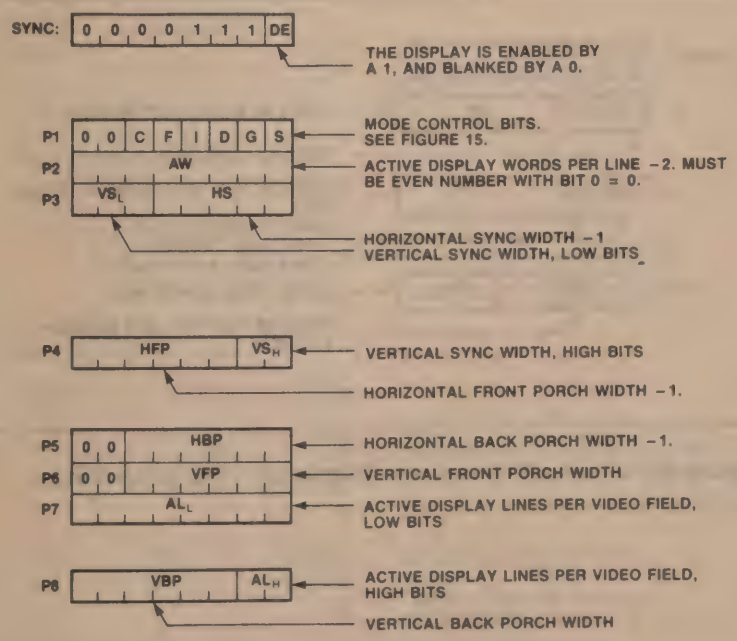


Figure 16. Sync Command

SYNC Format Specify Command

This command loads parameters into the sync generator. The various parameter fields and bits are identical to those at the RESET command. The GDC is not reset nor does it enter idle mode.

Vertical Sync Mode Command

When using two or more GDCs to contribute to one image, one GDC is defined as the master sync generator, and the others operate as its slaves. The VSYNC pins of all GDCs are connected together.

Slave Mode Operation

A few considerations should be observed when synchronizing two or more GDCs to generate overlayed video via the VSYNC INPUT/OUTPUT pin. As mentioned above, the Horizontal Front Porch (HFP) must be 4 or more display cycles wide. This is equivalent to eight or more clock cycles. This gives the slave GDCs time to initialize their internal video sync generators to the proper point in the video field to match the incoming vertical sync pulse (VSYNC). This resetting of the generator occurs just after the end of the incoming VSYNC pulse, during the HFP interval. Enough time during HFP is required to allow the slave GDC to complete the operation before the start of the HSYNC interval.

Once the GDCs are initialized and set up as Master and Slaves, they must be given time to synchronize. It is a good idea to watch the VSYNC status bit of the Master GDC and wait until after one or more VSYNC pulses have been generated before the display process is started. The START command will begin the active display of data and will end the video synchronization process, so be sure there has been at least one VSYNC pulse generated for the Slaves to synchronize to.

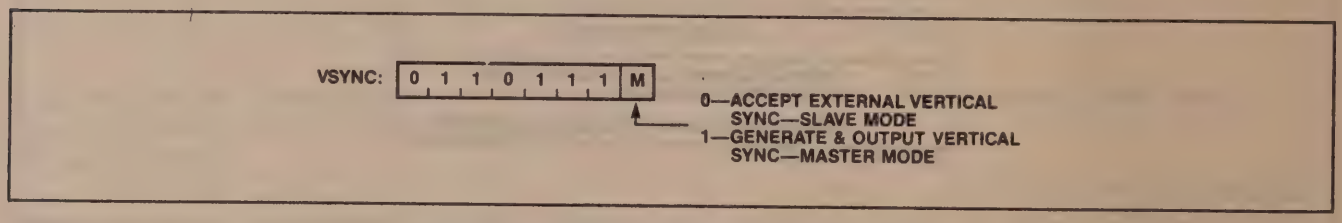


Figure 17. Vertical Sync Mode Command

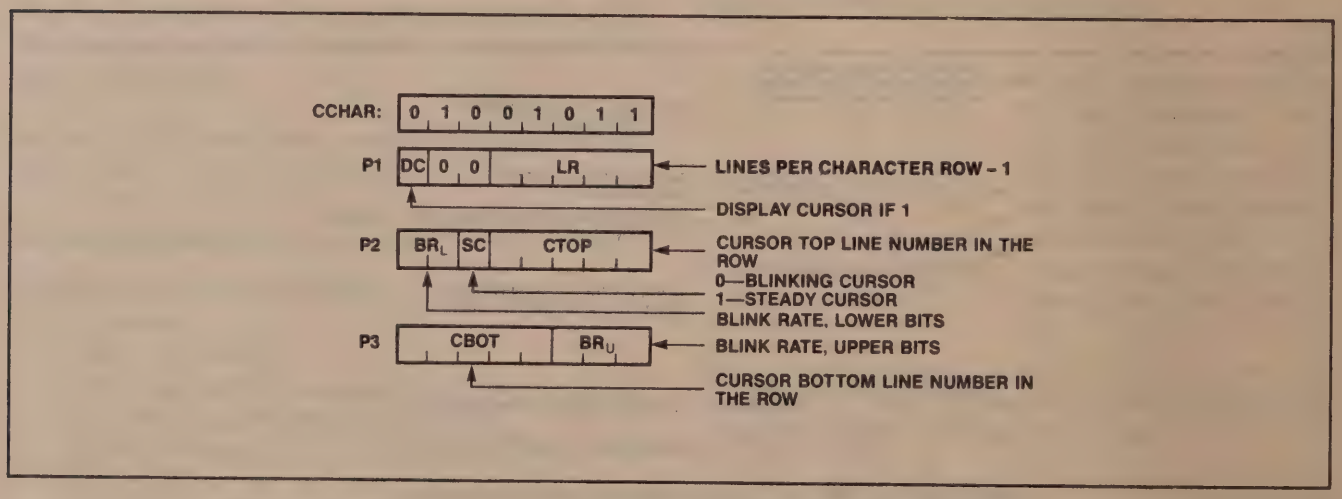


Figure 18. Cursor & Character Characteristics Command

Cursor and Character Characteristics Command

In graphics mode, LR should be set to 0. For interlaced displays in graphics mode, BR should be set to 3. The blink rate parameter controls both the cursor and attribute blink rates. The cursor blink-on-time = blink-off-time = 2 × BR (video frames). The attribute blink rate is always ½ the cursor rate but with a ¾ on-¼ off duty cycle.

DISPLAY CONTROL COMMANDS

Zoom Factors Specify Command

Zoom magnification factors of 1 through 16 are available using codes 0 through 15, respectively.

Cursor Position Specify Command

In character mode, the third parameter byte is not needed. The cursor is displayed for the word time in which the display scan address (DAD) equals the cursor address. In graphics mode, the cursor word address specifies the word containing the starting pixel of the drawing; the dot address value specifies the pixel within that word.

Parameter RAM Load Command

From the starting address, SA, any number of bytes may be loaded into the parameter RAM at incrementing addresses, up to location 15. The sequence of parameter bytes is terminated by the next command byte entered into the FIFO. The parameter RAM stores 16 bytes of information in predefined locations which differ for graphics and character modes. See the parameter RAM discussion for bit assignments.

Pitch Specification Command

This value is used during drawing by the drawing processor to find the word directly above or below the current word, and during display to find the start of the next line.

The Pitch parameter (width of display memory) is set by two different commands. In addition to the PITCH command, the RESET (or SYNC) command also sets the pitch value. The "active words per line" parameter, which specifies the width of the raster-scan display, also sets the Pitch of the display memory. In situations in which these two values are equal there is no need to execute a PITCH command.

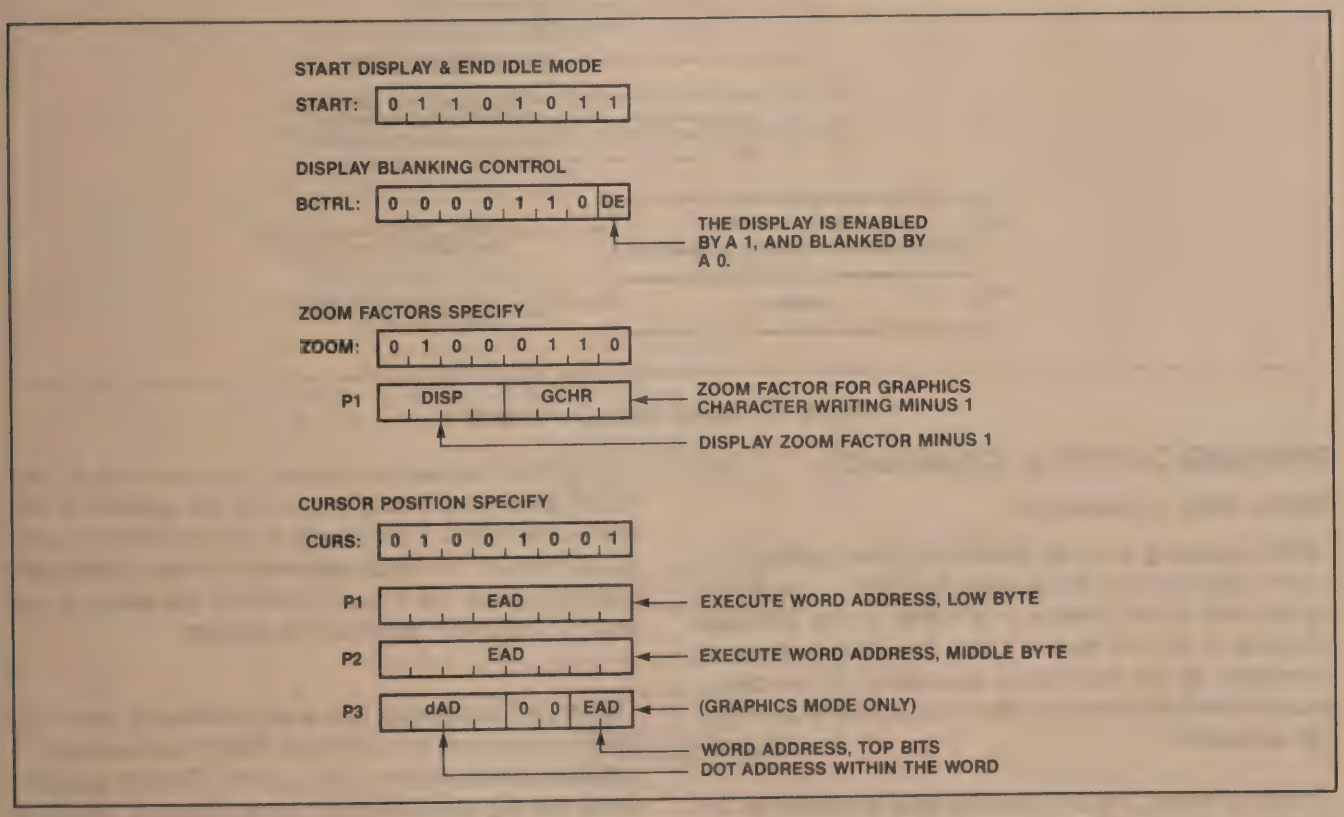


Figure 19. Display Control Commands

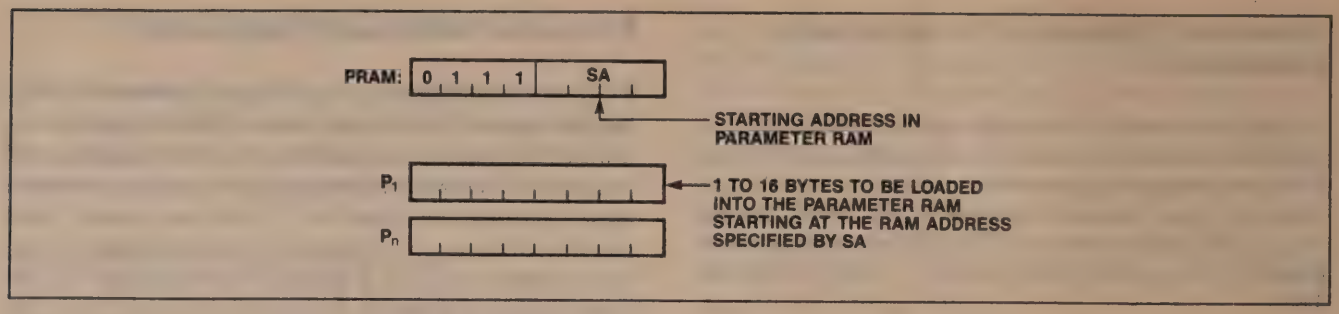


Figure 20. Parameter RAM Load Command

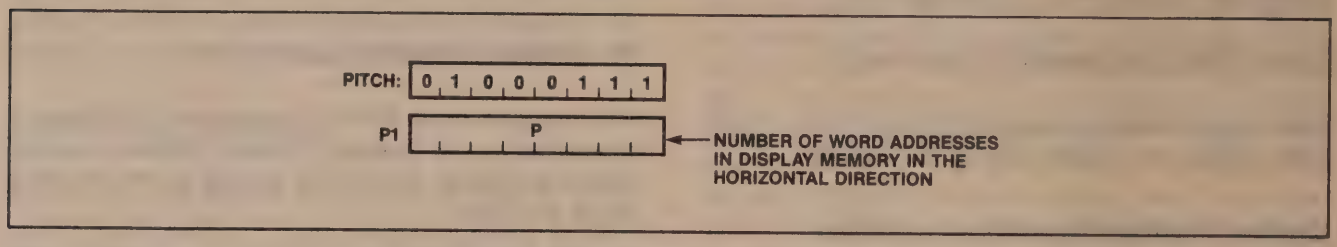


Figure 21. Pitch Specification Command

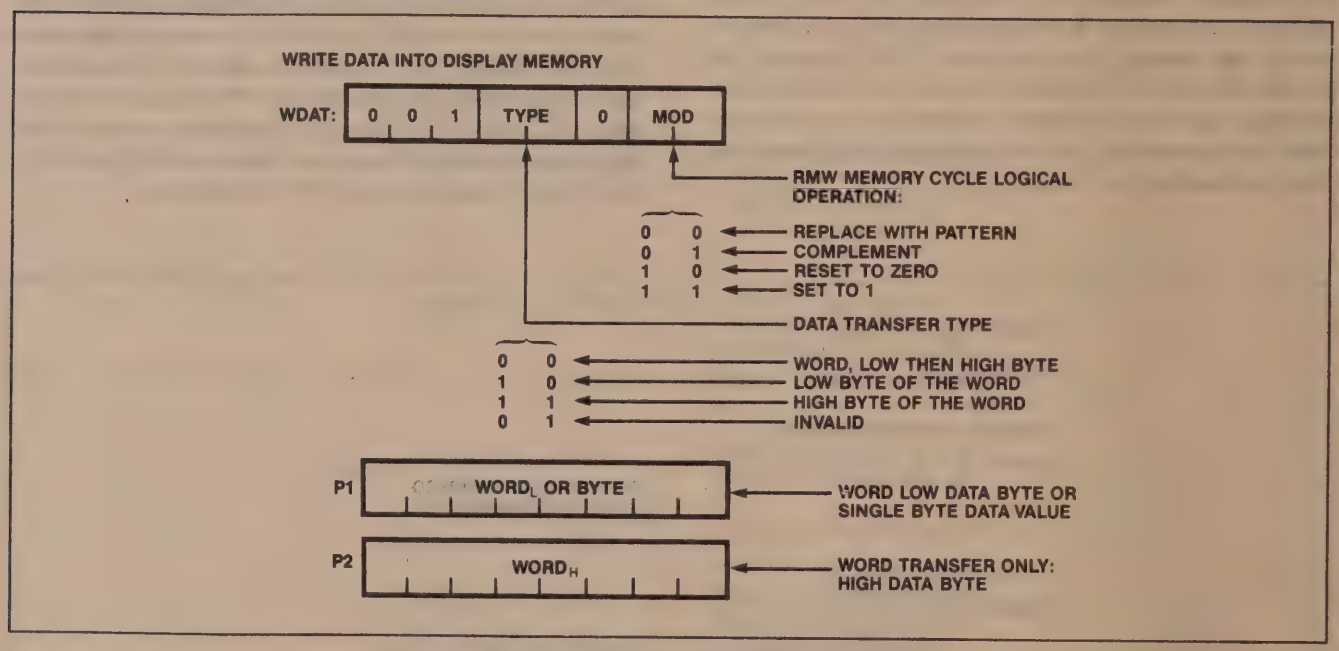


Figure 22. Write Data Command

DRAWING CONTROL COMMANDS

Write Data Command

Upon receiving a set of parameters (two bytes for a word transfer, one for a byte transfer), one RMW cycle into Video Memory is done at the address pointed to by the cursor EAD. The EAD pointer is advanced to the next word, according to the previously specified direction. More parameters can then be accepted.

For byte writes, the unspecified byte is treated as all zeros during the RMW memory cycle.

In graphics bit-map situations, only the LSB of the WDAT parameter bytes is used as the pattern in the RMW operations. Therefore it is possible to have only an all ones or all zeros pattern. In coded character applications all the bits of the WDAT parameters are used to establish the drawing pattern.

The WDAT command operates differently from the other commands which initiate RMW cycle activity. It requires parameters to set up the Pattern register while the other commands use the stored values in the parameter RAM. Like all of these commands, the

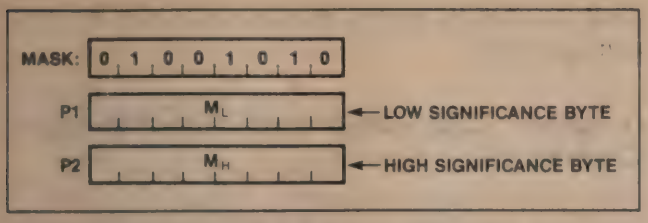
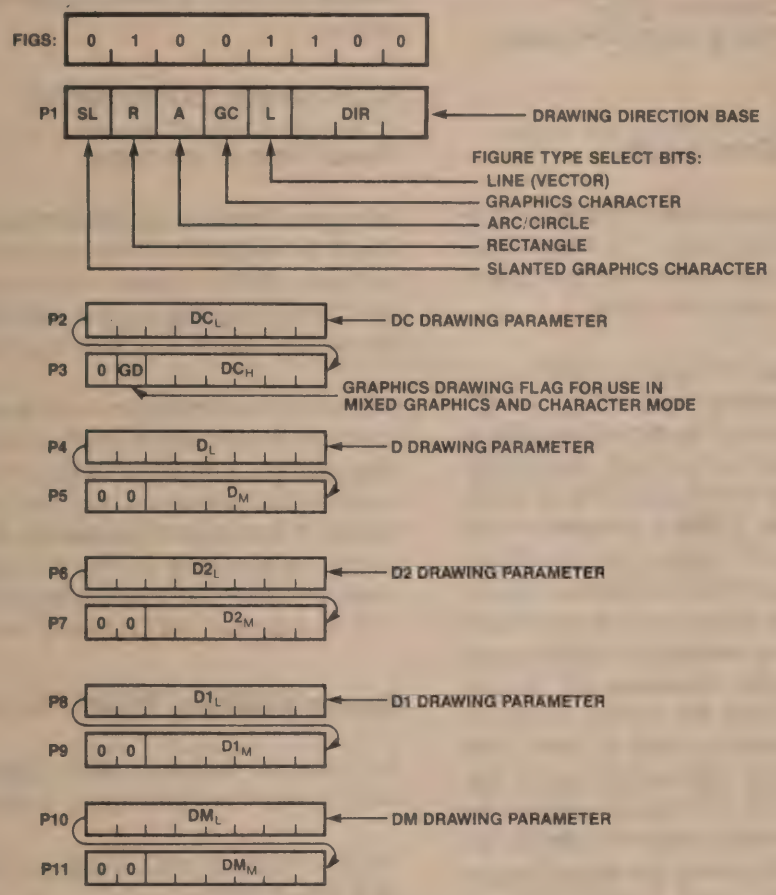


Figure 23. Mask Register Load Command

WDAT command must be preceded by a FIGS command and its parameters. Only the first three parameters need be given following the FIGS opcode, to set up the type of drawing, the DIR direction, and the DC value. The DC parameter +1 will be the number of RMW cycles done by the GDC with the first set of WDAT parameters. Additional sets of WDAT parameters will see a DC value of 0 which will cause only one RMW cycle to be executed.



VALID FIGURE TYPE SELECT COMBINATIONS

SL	R	A	GC	L	OPERATION
0	0	0	0	0	CHARACTER DISPLAY MODE DRAWING, INDIVIDUAL DOT DRAWING, DMA, WDAT, AND RDAT
0	0	0	0	1	STRAIGHT LINE DRAWING
0	0	0	1	0	GRAPHICS CHARACTER DRAWING AND AREA FILLING WITH GRAPHICS CHARACTER PATTERN
0	0	1	0	0	ARC AND CIRCLE DRAWING
0	1	0	0	0	RECTANGLE DRAWING
1	0	0	1	0	SLANTED GRAPHICS CHARACTER DRAWING AND SLANTED AREA FILLING

[ONLY THESE BIT COMBINATIONS ASSURE CORRECT DRAWING OPERATION]

Figure 24. Figure Drawing Parameters Specify Command

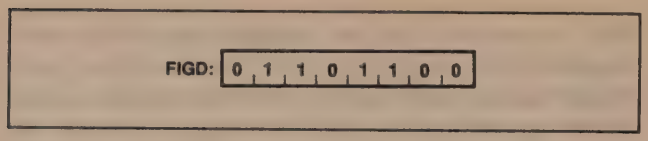


Figure 25. Figure Draw Start Command

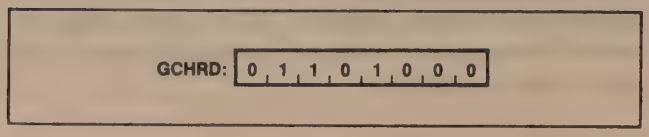


Figure 26. Graphics Character Draw and Area Filling Start Command

Mask Register Load Command

This command sets the value of the 16-bit Mask register of the figure drawing processor. The Mask register controls which bits can be modified in the display memory during a read-modify-write cycle.

The Mask register is loaded both by the MASK command and the third parameter byte of the CURS command. The MASK command accepts two parameter bytes to load a 16-bit value into the MASK register. All 16 bits can be individually one or zero, under program control. The CURS command on the other hand, puts a "1 of 16" pattern into the Mask register based on the value of the Dot Address value, dAD. If normal single-pixel-at-a-time graphics figure drawing is desired, there is no need to do a MASK command at all since the CURS command will set up the proper pattern to address the proper pixels as drawing progresses. For coded character DMA, and screen setting and clearing operations using the WDAT command, the MASK command should be used after the CURS command if its third parameter byte has been output. The Mask register should be set to all ones for any "word-at-a-time" operation.

Figure Draw Start Command

On execution of this instruction, the GDC loads the parameters from the parameter RAM into the drawing processor and starts the drawing process at the pixel pointed to by the cursor, EAD, and the dot address, dAD.

Graphics Char. Draw and Area Fill Start Command

Based on parameters loaded with the FIGS command, this command initiates the drawing of the graphics character or area filling pattern stored in Parameter RAM. Drawing begins at the address in display memory pointed to by the EAD and dAD values.

DATA READ COMMANDS

Read Data Command

Using the DIR and DC parameters of the FIGS command to establish direction and transfer count, multiple RMW cycles can be executed without specification of the cursor address after the initial load (DC = number of words or bytes).

As this instruction begins to execute, the FIFO buffer direction is reversed so that the data read from display memory can pass to the microprocessor. Any commands or parameters in the FIFO at this time will be lost. A command byte sent to the GDC will immediately reverse the buffer direction back to write mode, and all RDAT information not yet read from the FIFO will be lost. MOD should be set to 00.

Cursor Address Read Command

The Execute Address, EAD, points to the display memory word containing the pixel to be addressed.

The Dot Address, dAD, within the word is represented as a 1-of-16 code.

Light Pen Address Read Command

The light pen address, LAD, corresponds to the display word address, DAD, at which the light pen input signal is detected and deglitched.

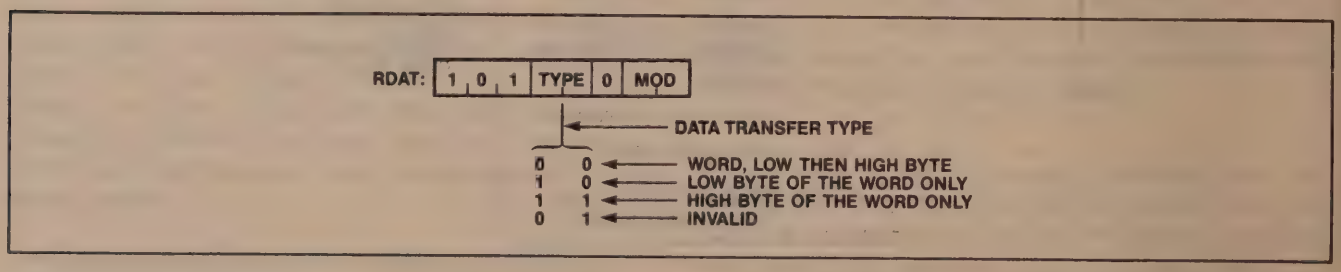


Figure 27. Read Data from Display Memory Command

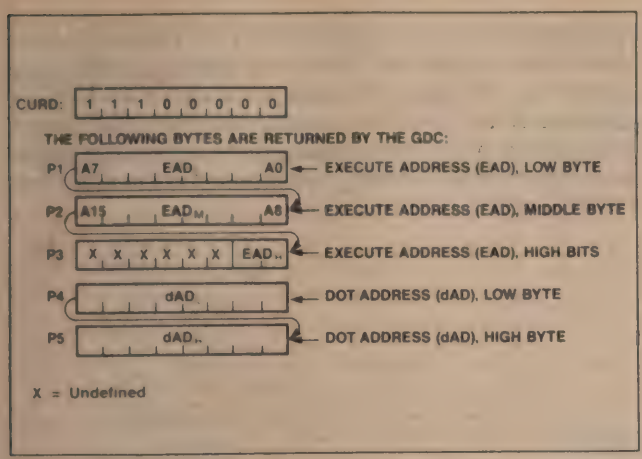


Figure 28. Cursor Address Read Command

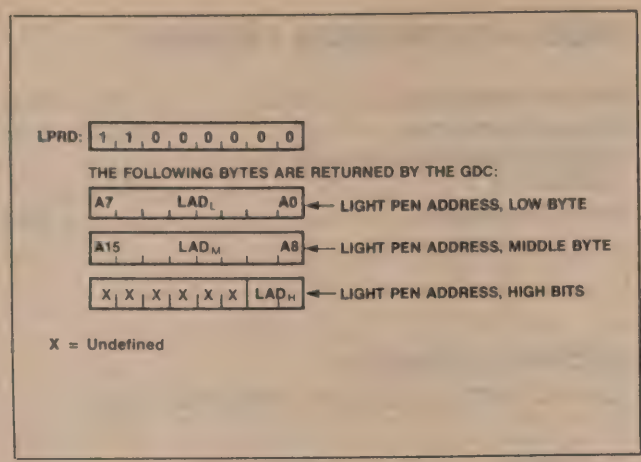


Figure 29. Light Pen Address Read Command

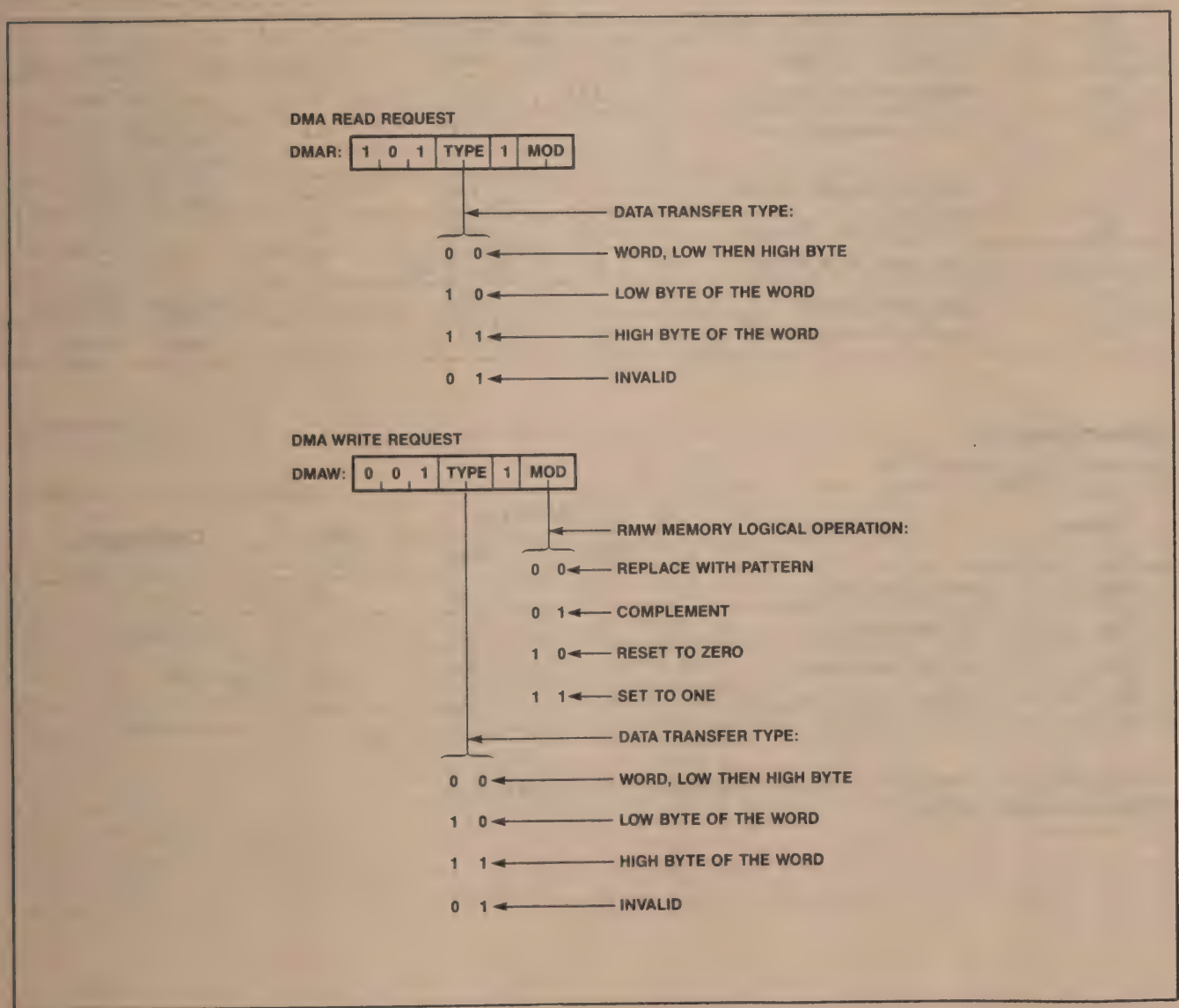


Figure 30. DMA Control Commands

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias ..... 0°C to 70°C
Storage Temperature ..... −65°C to 150°C
Voltage on any Pin with Respect to Ground ..... −0.5V to +7V
Power Dissipation ..... 1.5 Watt

**COMMENT: Exposing the device to stresses above those listed in Absolute Maximum Ratings could cause permanent damage. The device is not meant to be operated under conditions outside the limits described in the operational sections of this specification. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

DC CHARACTERISTICS

T_A = 0°C to 70° C; V_{CC} = 5V ± 10%; GND = 0V

Symbol	Parameter	Limits		Unit	Conditions
		Min.	Max.		
V_{IL}	Input Low Voltage	−0.5	0.8	V	
V_{IH}	Input High Voltage Except $\overline{DACK}$	2.2	V_{CC} + 0.5	V	
V_{OL}	Output Low Voltage		0.45	V	I_{OL} = 2.2 mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = −400 μA
I_{OZ}	I/O Pin Leakage Current		±10	μA	V_{SS}+0.45 ≤ V_I ≤ V_{CC}
I_{IL}	Input Pin Leakage Current		±10	μA	V_{SS} ≤ V_I ≤ V_{CC}
V_{CL}	Clock Input Low Voltage	−0.5	0.6	V	
V_{CH}	Clock Input High Voltage	3.5	V_{CC} + 0.5	V	
I_{CC}	V_{CC} Supply Current		270	mA	Typical = 150 mA
VIH1	Input High Voltage $\overline{DACK}$ Only	2.4	V_{CC} + 0.5	V	Typical = 150 mA

CAPACITANCE (2)

T_A = 25°C; V_{CC} = GND = 0V

Symbol	Parameter	Limits		Unit	Conditions
		Min.	Max.		
C_{IN}	Input Capacitance		10	pF	fc = 1 MHz V = 0
C_{IO}	I/O Capacitance		20	pF	
C_{OUT}	Output Capacitance		20	pF	
C_O	Clock Input Capacitance		20	pF	

(1) Suggest pull up resistor to reduce noise sensitivity on DACK only.
(2) Sample tested initially.

A.C. CHARACTERISTICS (T_A = 0°C to +70°C, V_{SS} = 0V, V_{CC} = +5V ±10%)

DATA BUS READ CYCLE

Symbol	Parameter	82720		82720-1		82720-2		Units	Test Conditions
		Min.	Max.	Min.	Max.	Min.	Max.		
T_{AR}	A_0 setup to $\overline{RD}$ ↓	0		0		0		ns	
T_{RA}	A_0 hold after $\overline{RD}$ ↑	0		0		0		ns	
T_{RR}	$\overline{RD}$ Pulse Width	$T_{RD}+20$		$T_{RD}+20$		$T_{RD}+20$		ns	
T_{RD}	$\overline{RD}$↓ to Data Out Delay		120		80		70	ns	CL = 50pF
T_{DF}	$\overline{RD}$↑ to Data Float Delay	0	120	0	100	0	90	ns	
T_{RV}	$\overline{RD}$ Recovery Time	4 T_{CY}		4 T_{CY}		4 T_{CY}		ns	

DATA BUS WRITE CYCLE

Symbol	Parameter	82720		82720-1		82720-2		Units	Test Conditions
		Min.	Max.	Min.	Max.	Min.	Max.		
T_{AW}	A_0 Setup to $\overline{WR}$ ↓	0		0		0		ns	
T_{WA}	A_0 Hold after $\overline{WR}$ ↑	0		0		10		ns	
T_{WW}	$\overline{WR}$ Pulse Width	120		100		90		ns	
T_{DW}	Data Setup to $\overline{WR}$ ↑	100		80		70		ns	
T_{WD}	Data Hold after $\overline{WR}$ ↑	10		10		10		ns	
T_{RV}	$\overline{WR}$ Recovery Time	4 T_{CY}		4 T_{CY}		4 T_{CY}		ns	

DISPLAY MEMORY TIMING

Symbol	Parameter	82720		82720-1		82720-2		Units	Test Conditions
		Min.	Max.	Min.	Max.	Min.	Max.		
T_{CA}	Address/Data Delay from 2XWCLK ↑	30	160	30	130	30	110	ns	CL = 50pF
T_{AC}	Address/Data Hold Time	30	160	30	130	30	110	ns	CL = 50pF
T_{DC}	Input Data Setup to 2XWCLK ↓	0		0		0		ns	
T_{CD}	Input Data Hold Time	T_{IE}		T_{IE}		T_{IE}		ns	
T_{IE}	2XWCLK ↑ to $\overline{DBIN}$	30	120	30	90	30	80	ns	CL = 50pF
T_{CAH}	2XWCLK ↑ to ALE ↑	30	125	30	100	30	90	ns	CL = 50pF
T_{AL}	ALE Low Time	$T_{CY}+30$		$T_{CY}+30$		$T_{CY}+30$		ns	
T_{AH}	ALE High Time	$T_{CH}-20$		$T_{CH}-20$		$T_{CH}-20$		ns	
T_{CO}	Video Signal Delay from 2XWCLK ↑		150		120		100	ns	
T_{LLAX}	Address Valid Hold Time After ALE.	30		30		30		ns	
T_{AVAL}	Address Valid Hold Time Before ALE.	20		10		5		ns	

A.C. CHARACTERISTICS (Continued)

OTHER TIMING

Symbol	Parameter	82720		82720-1		82720-2		Units	Test Conditions
		Min.	Max.	Min.	Max.	Min.	Max.		
T_{PC}	LPEN or VSYNC Input Setup to 2XWCLK↑	30		20		15		ns	
T_{PP}	LPEN or VSYNC Input Pulse Width	T_{CY}		T_{CY}		T_{CY}		ns	

CLOCK TIMING

Symbol	Parameter	82720		82720-1		82720-2		Units	Test Conditions
		Min.	Max.	Min.	Max.	Min.	Max.		
T_{CY}	Clock Period	250	2000	200	2000	180	2000	ns	
T_{CH}	Clock High Time	105		90		70		ns	
T_{CL}	Clock Low Time	105		80		70		ns	
T_R	Rise Time		20		20		20	ns	
T_F	Fall Time		20		20		20	ns	

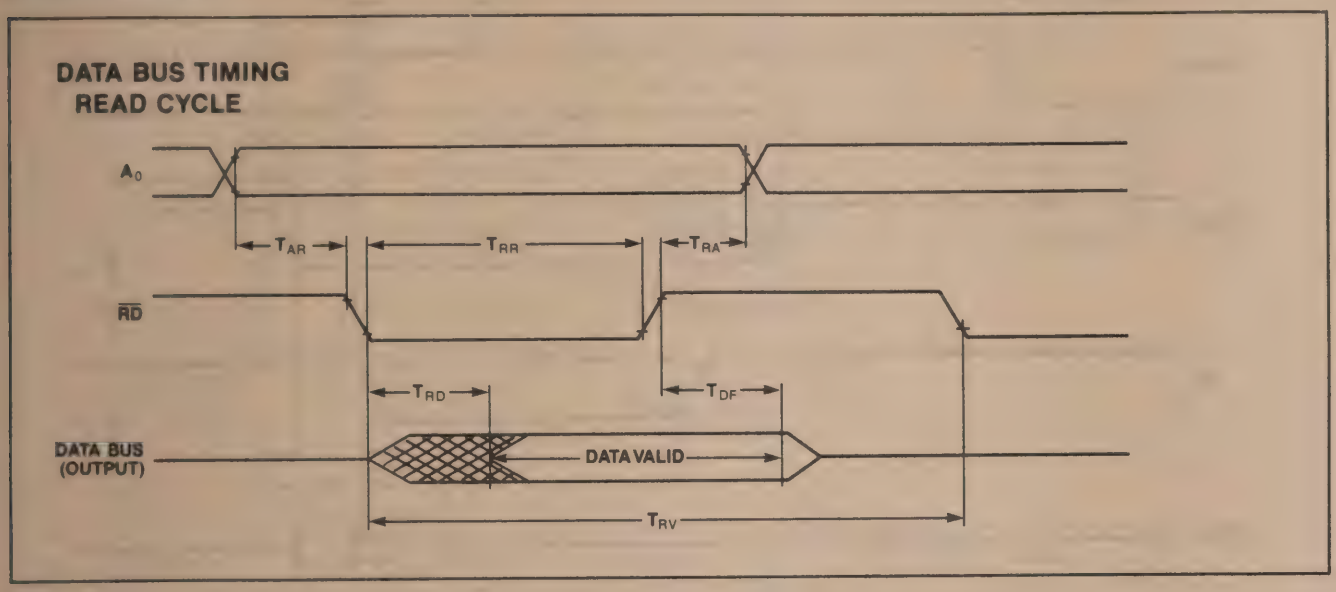
DMA TIMING

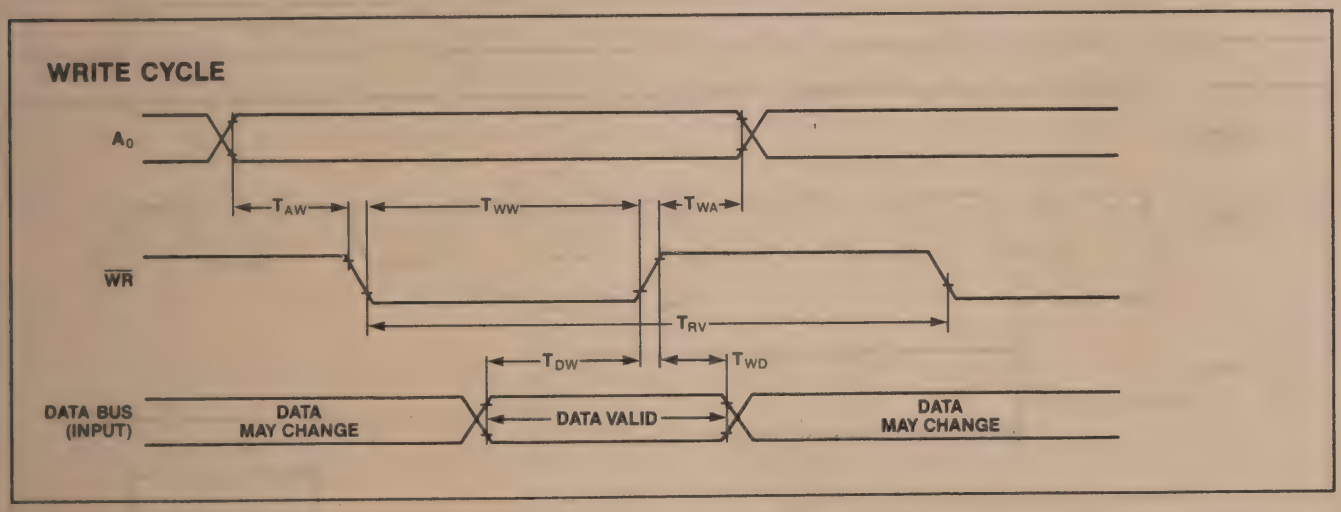
Symbol	Parameter	82720		82720-1		82720-2		Units	Test Conditions
		Min.	Max.	Min.	Max.	Min.	Max.		
T_{ACC}	$\overline{DACK}$ Setup to $\overline{RD}$↓ or $\overline{WR}$↓	0		0		0		ns	
T_{CAC}	$\overline{DACK}$ Hold from $\overline{RD}$↑ or $\overline{WR}$↑	0		0		0		ns	
T_{RR1}	$\overline{RD}$ Pulse Width	$T_{RD1}+20$		$T_{RD1}+20$		$T_{RD1}+20$		ns	
T_{RD1}	$\overline{RD}$↓ to Data Out Delay		$1.5\ T_{CY}$ $+120$		$1.5\ T_{CY}$ $+80$		$1.5\ T_{CY}$ $+70$	ns	CL = 50pF
T_{KQ}	2XWCLK↑ to DREQ Delay		150		120		100	ns	CL = 50pF
T_{RQAK}	DREQ Setup to DACK↓	0		0		0		ns	
T_{AKRQ}	$\overline{DACK}$↓ to DREQ↓ Delay		$T_{CY}+150$		$T_{CY}+120$		$T_{CY}+100$	ns	CL = 50pF
T_{AKH}	$\overline{DACK}$ High Time	T_{CY}		T_{CY}		T_{CY}		ns	
T_{AK1}	$\overline{DACK}$ Cycle Time, Word Mode	$4\ T_{CY}$		$4\ T_{CY}$		$4\ T_{CY}$		ns	
T_{AK2}	$\overline{DACK}$ Cycle Time, Byte Mode	$5\ T_{CY}$		$5\ T_{CY}$		$5\ T_{CY}$		ns	

A.C. TEST CONDITIONS

Input Pulse Levels (except 2XWCLK) ..... 0.45V to 2.4V
Input Pulse Levels (2XWCLK) ..... 0.3V to 3.9V
Timing Measurement Reference Levels (except 2XWCLK) ..... 0.8V to 2.0V
Timing Measurement Reference Levels (2XWCLK) ..... 0.6V to 3.5V

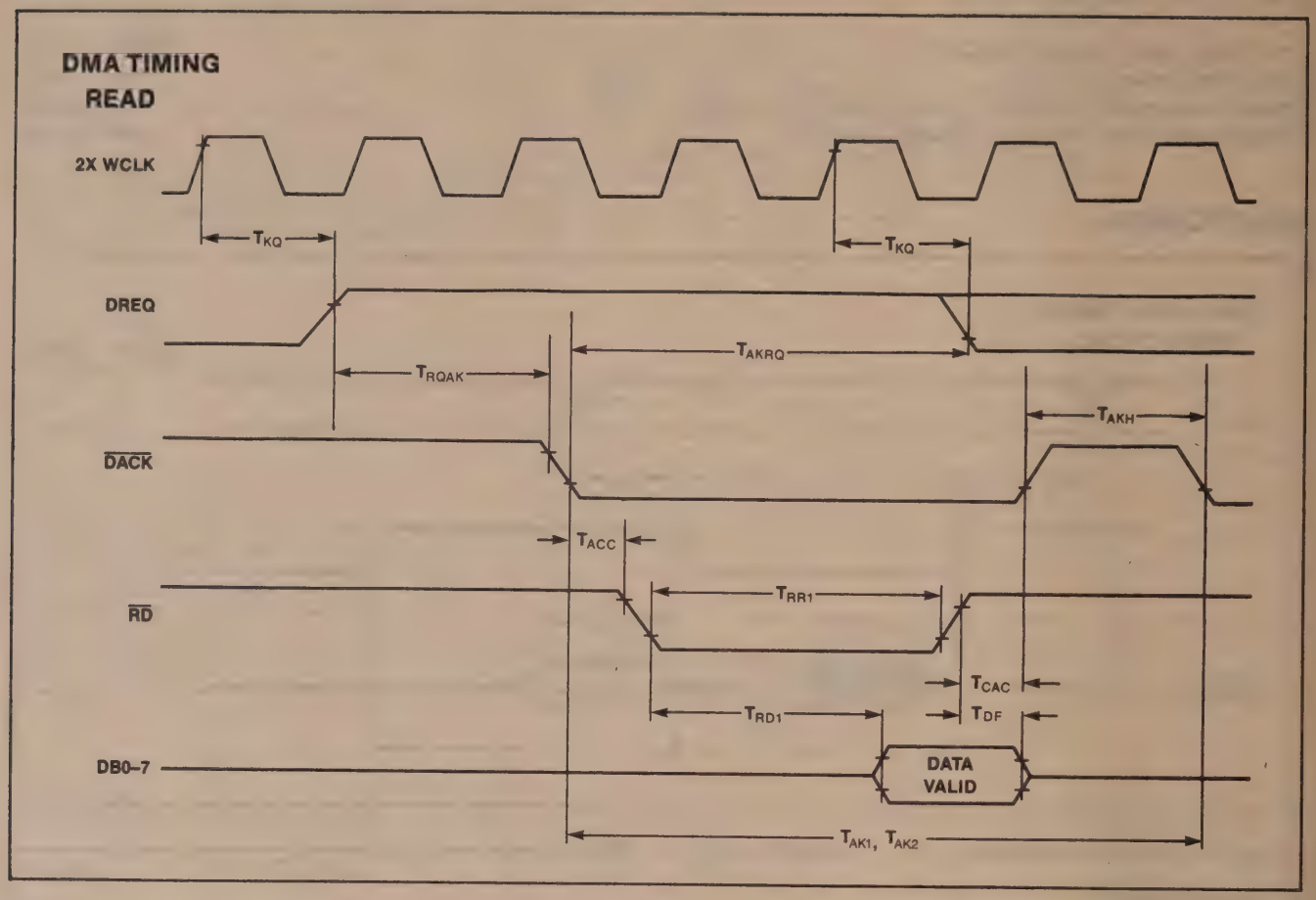
WAVEFORMS

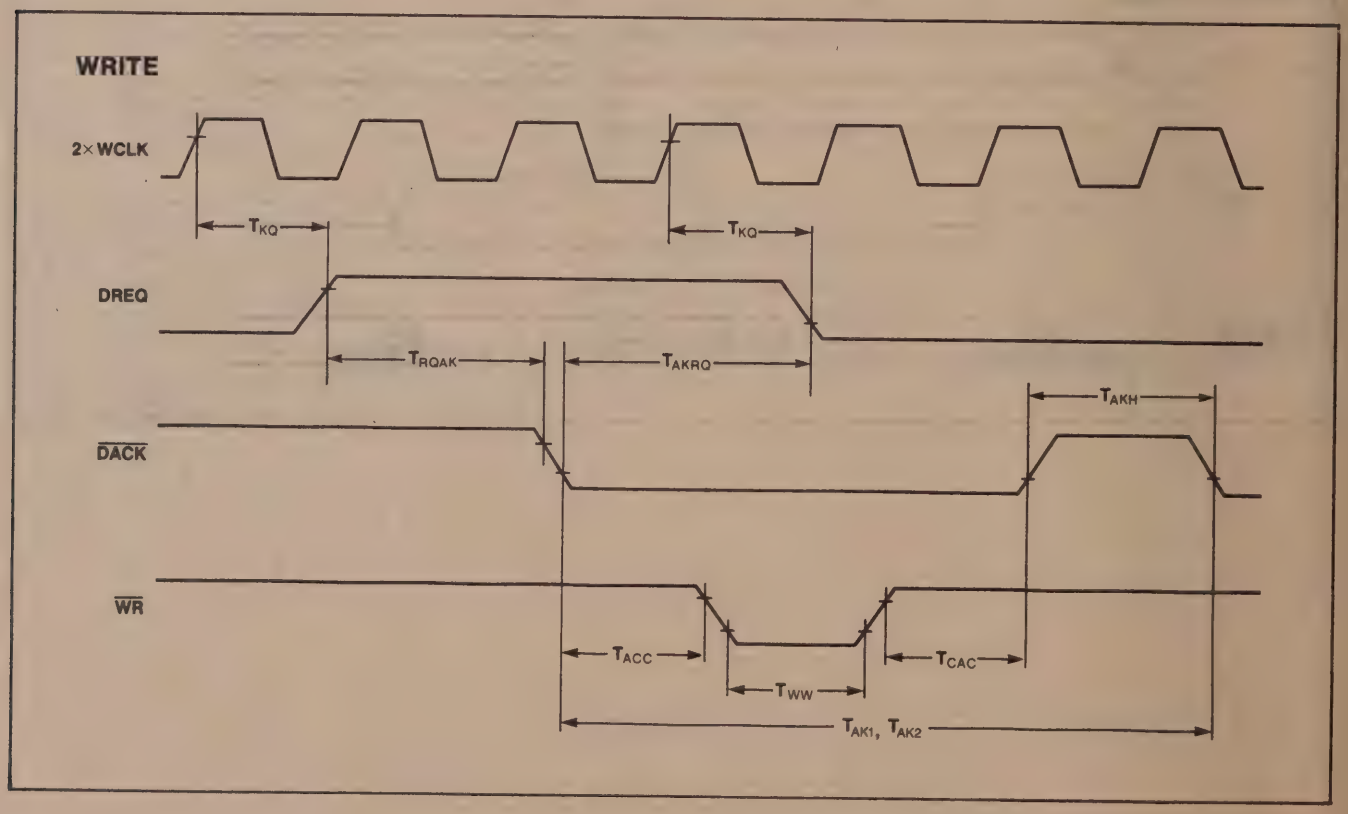




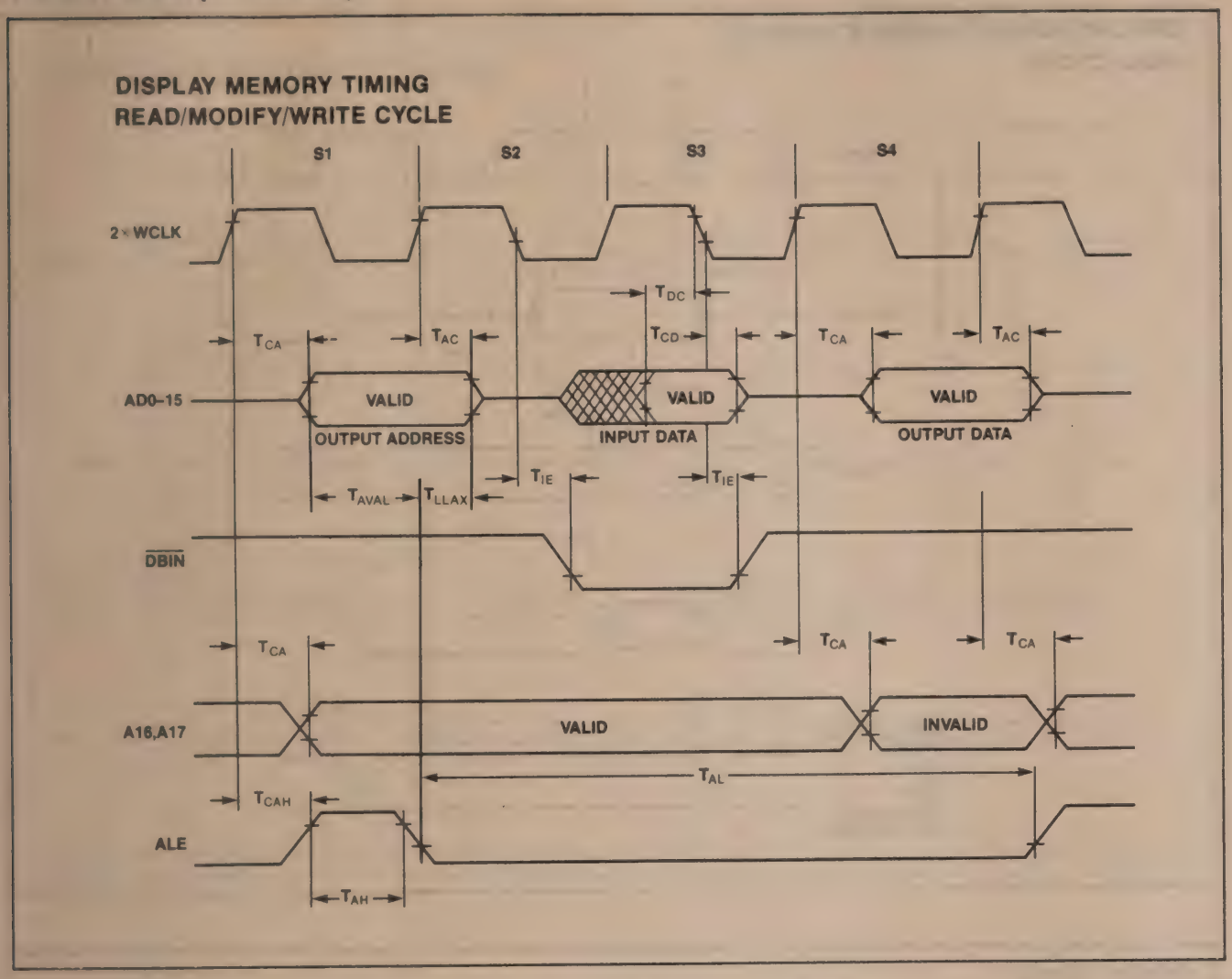
210655-003

WAVEFORMS (Continued)





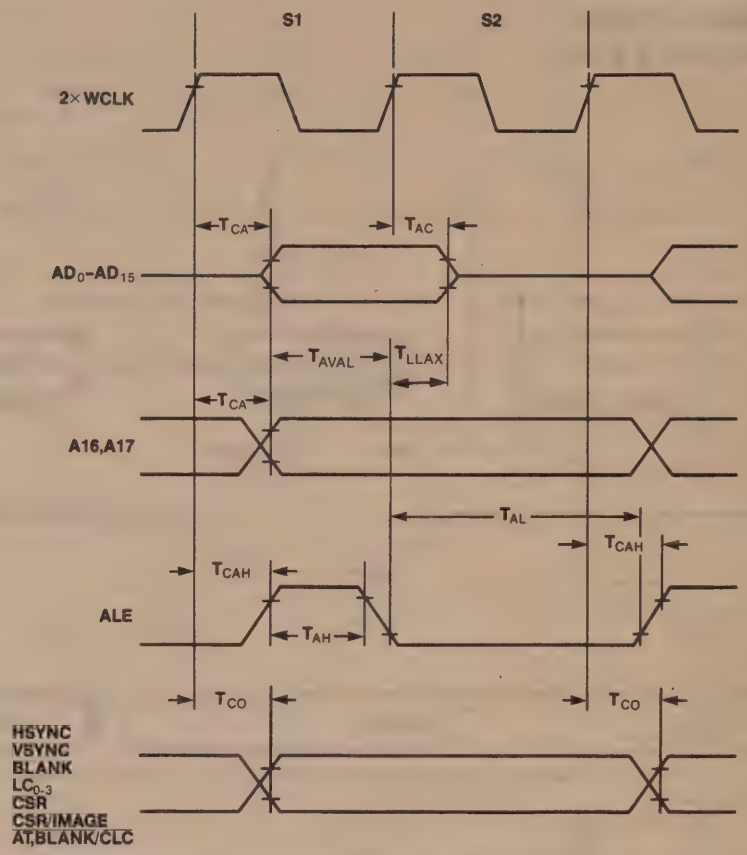
WAVEFORMS (Continued)



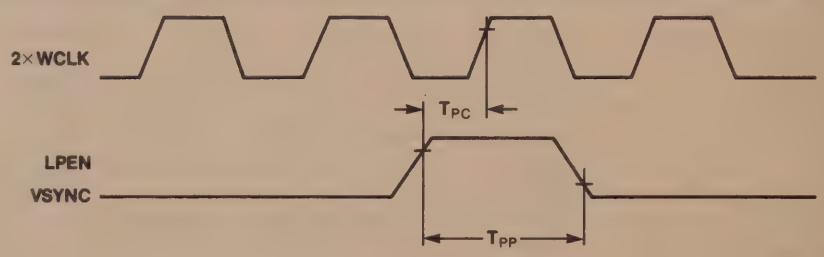
WAVEFORMS (Continued)

DISPLAY MEMORY TIMING (Continued)

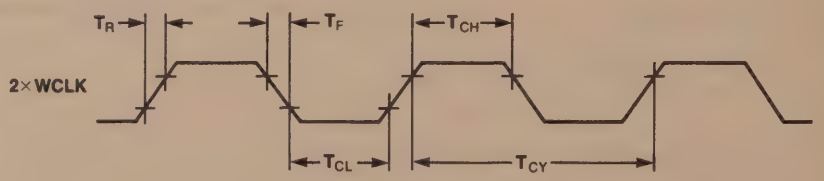
READ CYCLE



OTHER TIMING

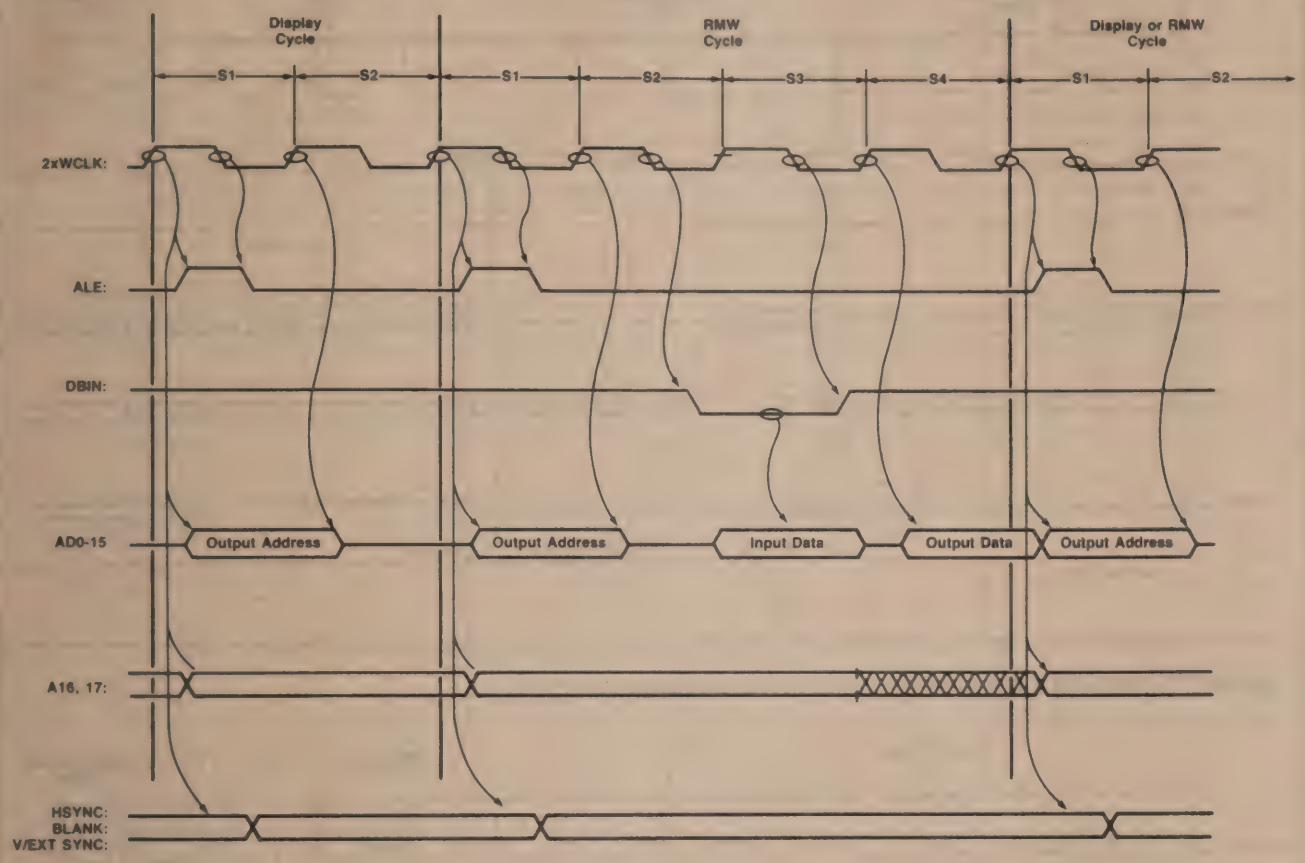


CLOCK TIMING



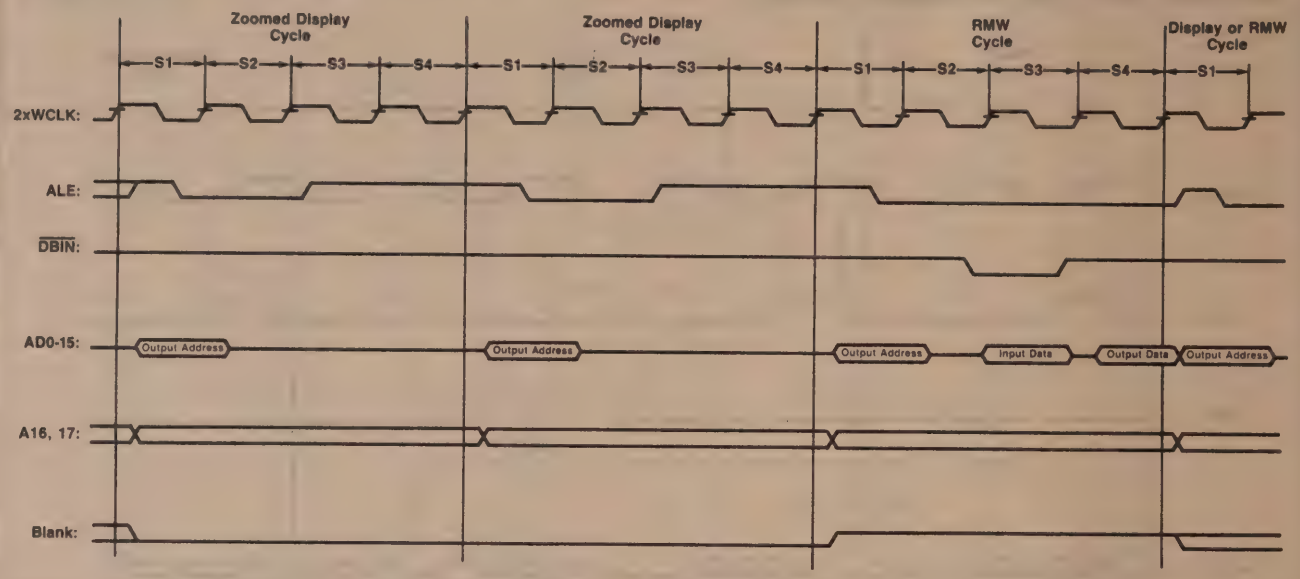
WAVEFORMS (Continued)

DISPLAY AND RMW CYCLES (1X ZOOM)

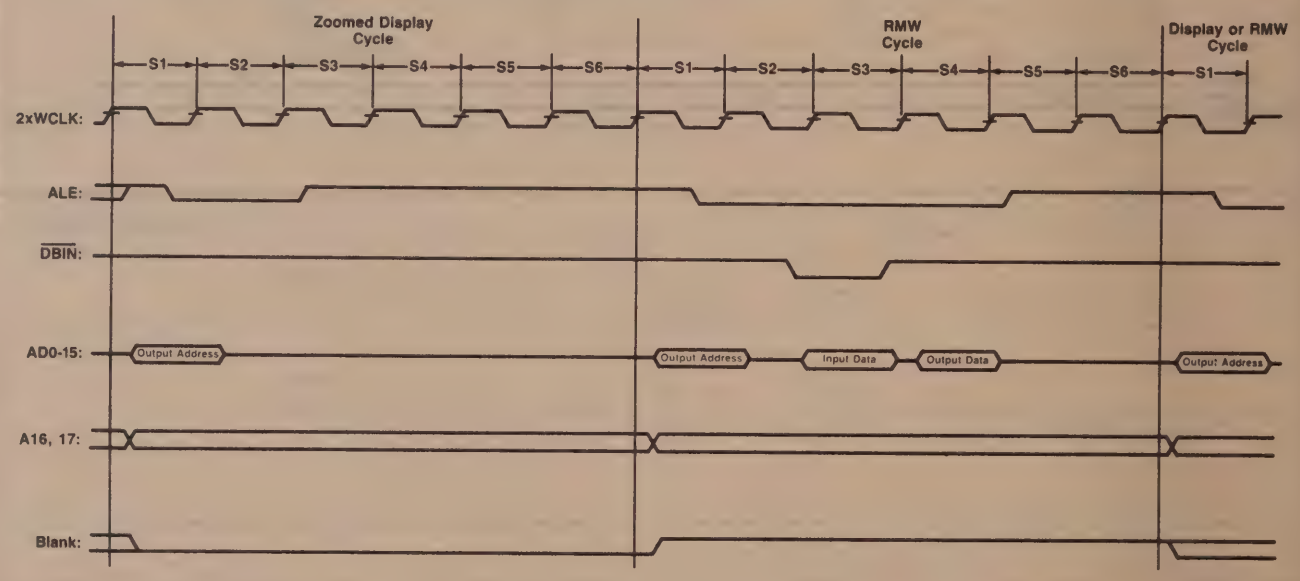


WAVEFORMS (Continued)

DISPLAY AND RMW CYCLES (2X ZOOM)



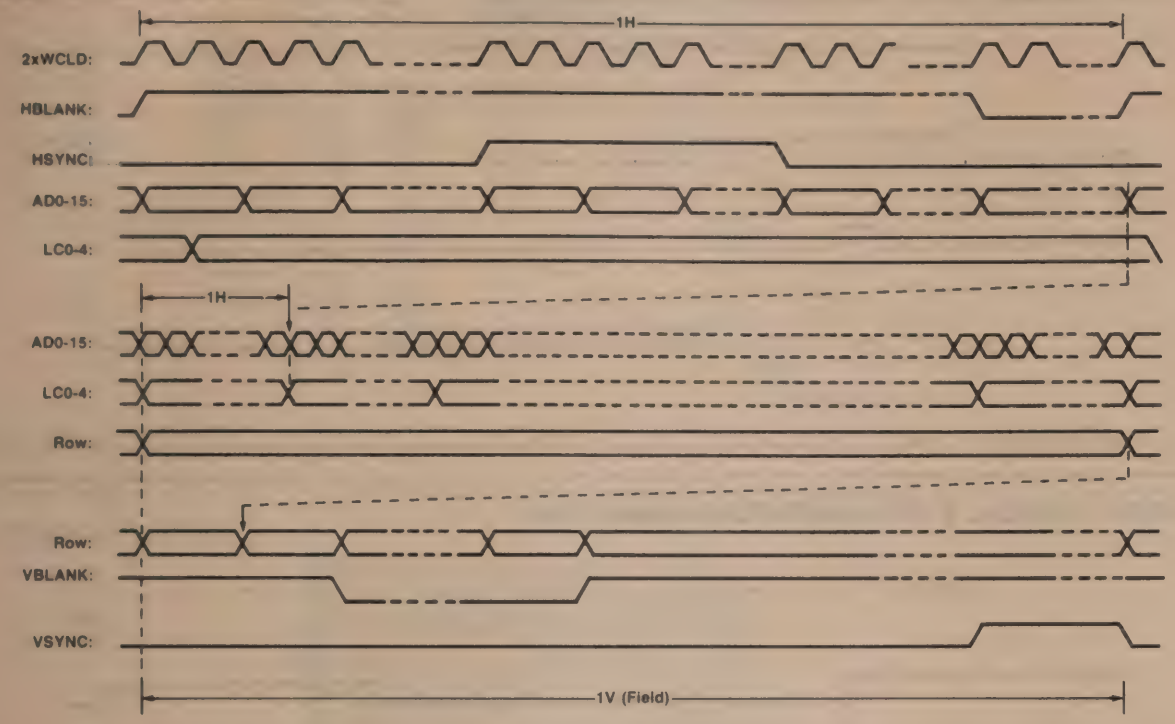
ZOOMED DISPLAY OPERATION WITH RMW CYCLE (3X ZOOM)



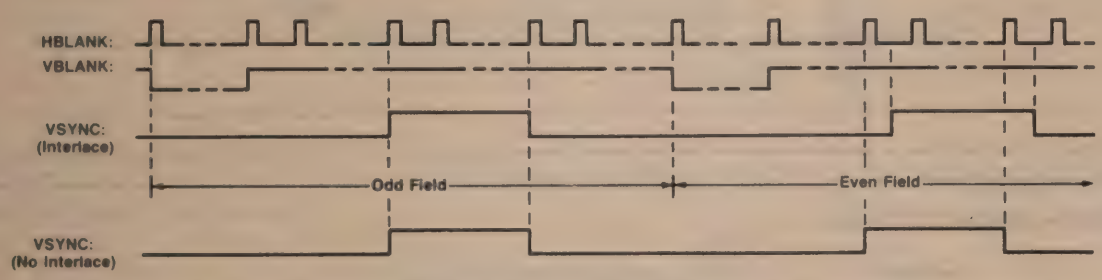
210655-003

WAVEFORMS (Continued)

VIDEO SYNC SIGNALS TIMING

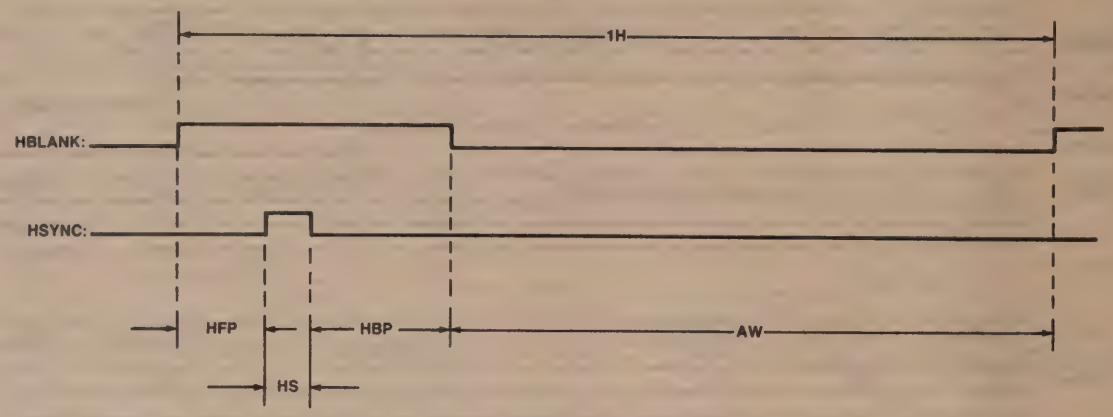


INTERLACED VIDEO TIMING

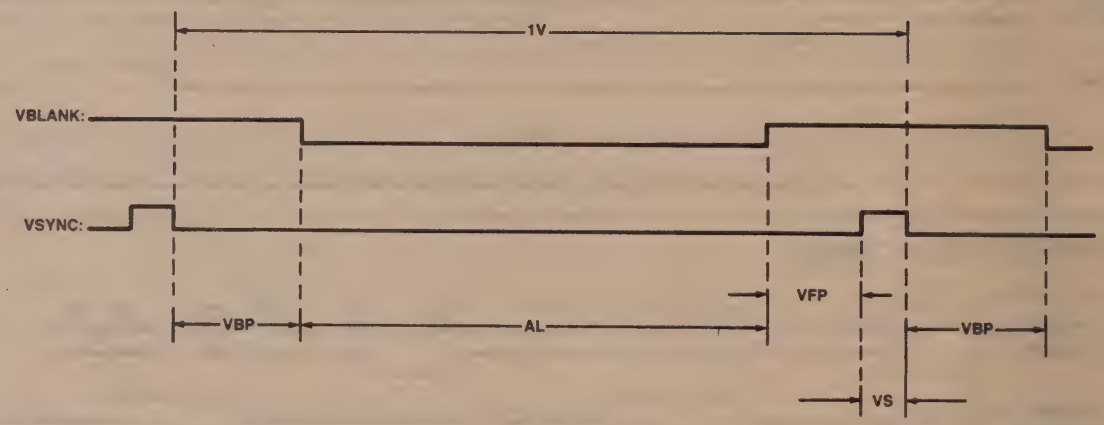


WAVEFORMS (Continued)

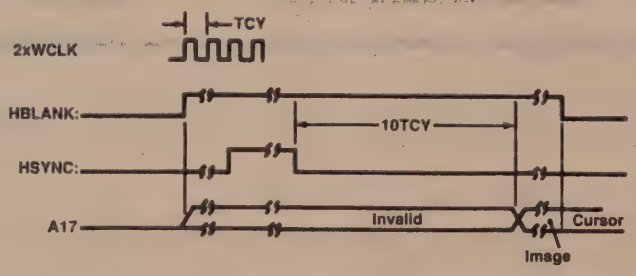
VIDEO HORIZONTAL SYNC GENERATOR PARAMETERS



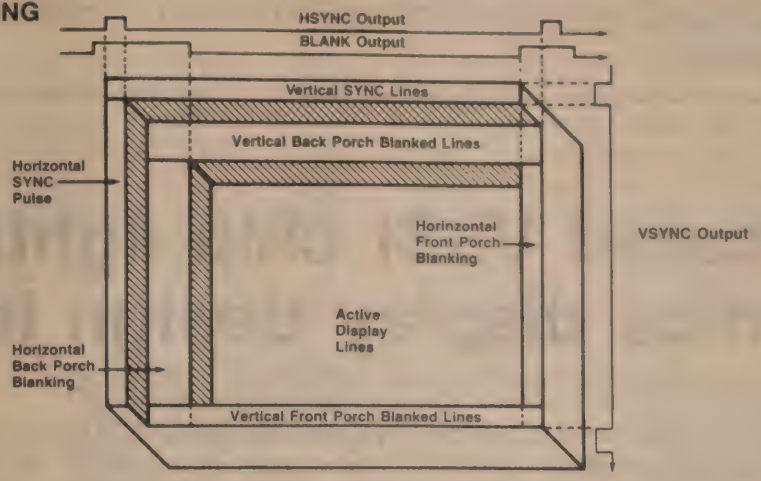
VIDEO VERTICAL SYNC GENERATOR PARAMETERS



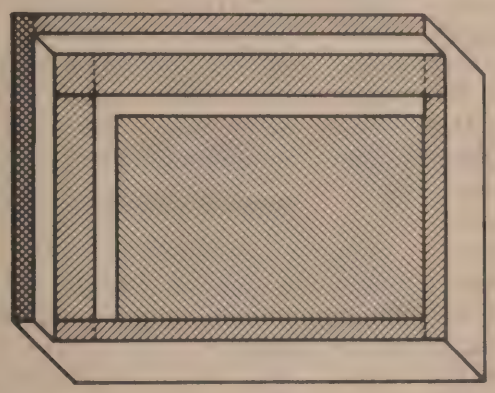
CURSOR—IMAGE BIT FLAG



VIDEO FIELD TIMING



DRAWING INTERVALS

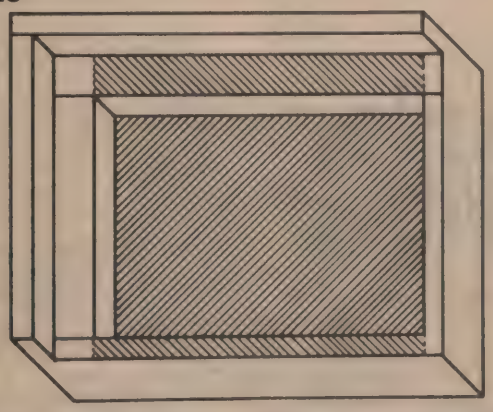


Drawing Interval

Additional Drawing Interval When in Flash Mode

Dynamic RAM Refresh If Enabled, Otherwise Additional Drawing Interval

DMA REQUEST INTERVALS



DMA Request Interval

Additional DMA Request Intervals When in Flash Mode

By managing tasks like graphics generation and CRT refreshing, a dedicated VLSI display controller simplifies the design of intelligent graphics work stations.

Dedicated VLSI chip lightens graphics display design load

The role of graphics is becoming increasingly important for unscrambling the communications traffic between people and computers. Thanks to microprocessors and dedicated control ICs, designing high-reliability graphics work stations is now easier and less expensive than in the days of small-scale integration and expensive discrete-circuit CRT technology. Microprocessors simplify work-station design by transferring some graphics control tasks from hardware to software. However, a dedicated VLSI controller such as the 82720—with an on-board graphics processor—can push another step forward toward fast and economical design of high-quality intelligent graphics systems.

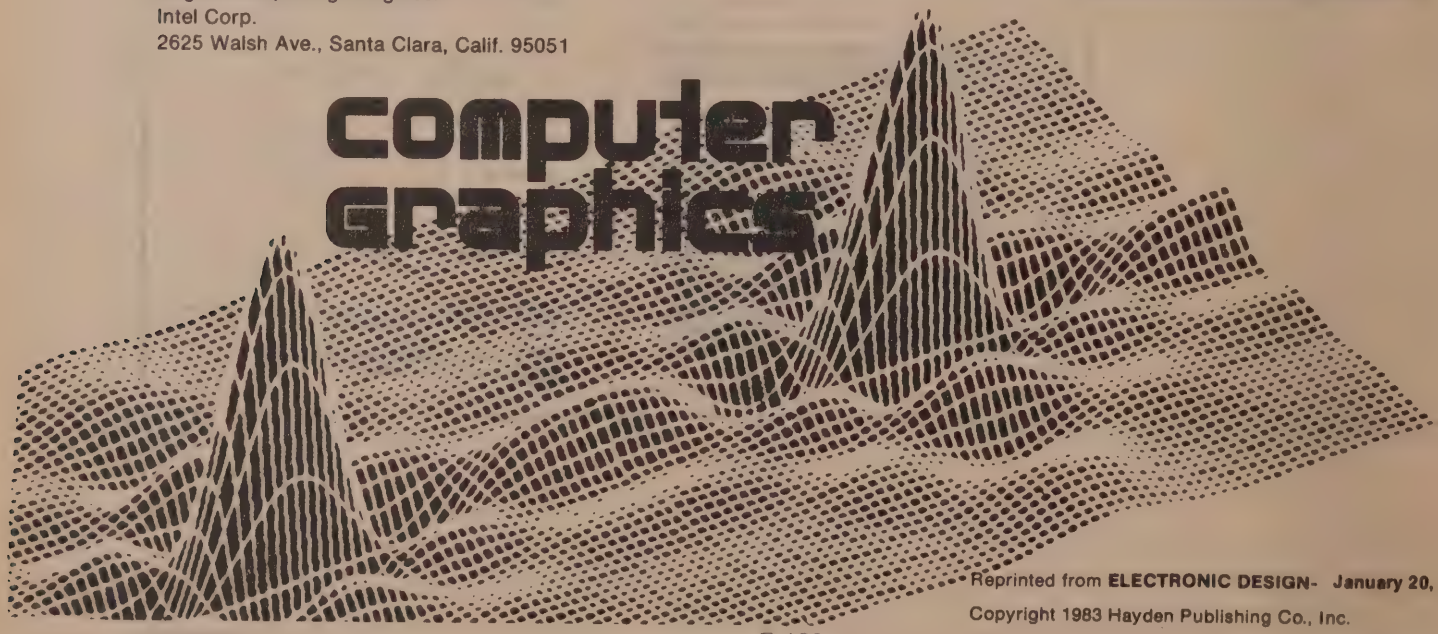
A typical application for the controller is a graphics work station aimed at high-end business and low-end engineering systems. Since such a station usually fits on the top of a desk, all of the electronics must be contained within a single printed-circuit board. This type of system requires a resolution of about 512 by 512 pixels and is frequently called on to display three-dimensional objects in various perspectives. To minimize the distortion of rotating objects, horizontal and vertical pixels should be equally spaced.

A typical display (500 vertices) must be drawn on the screen in less than 1 second to provide satisfactory interaction with the operator. The display may consist of lines, arcs, filled areas, and colors—seven colors are acceptable (see "A Look into Graphics Fundamentals").

Serial link interfaces station

An intelligent work station usually interfaces with a mainframe host via a serial communications link, a keyboard, and a serial link with an optional graphics tablet. This type of graphics input/output subsystem is diagrammed in Fig. 1. Two 5¼-in. floppy disks can satisfy the mass-storage needs of the system. Disk formatting must be compatible with the requirements of an IBM personal computer. Moreover, general-purpose software written for

Gary DePalma, Field Applications Engineer
Mark Olson, Product Marketing Engineer
Roger Jollis, Design Engineer
Intel Corp.
2625 Walsh Ave., Santa Clara, Calif. 95051



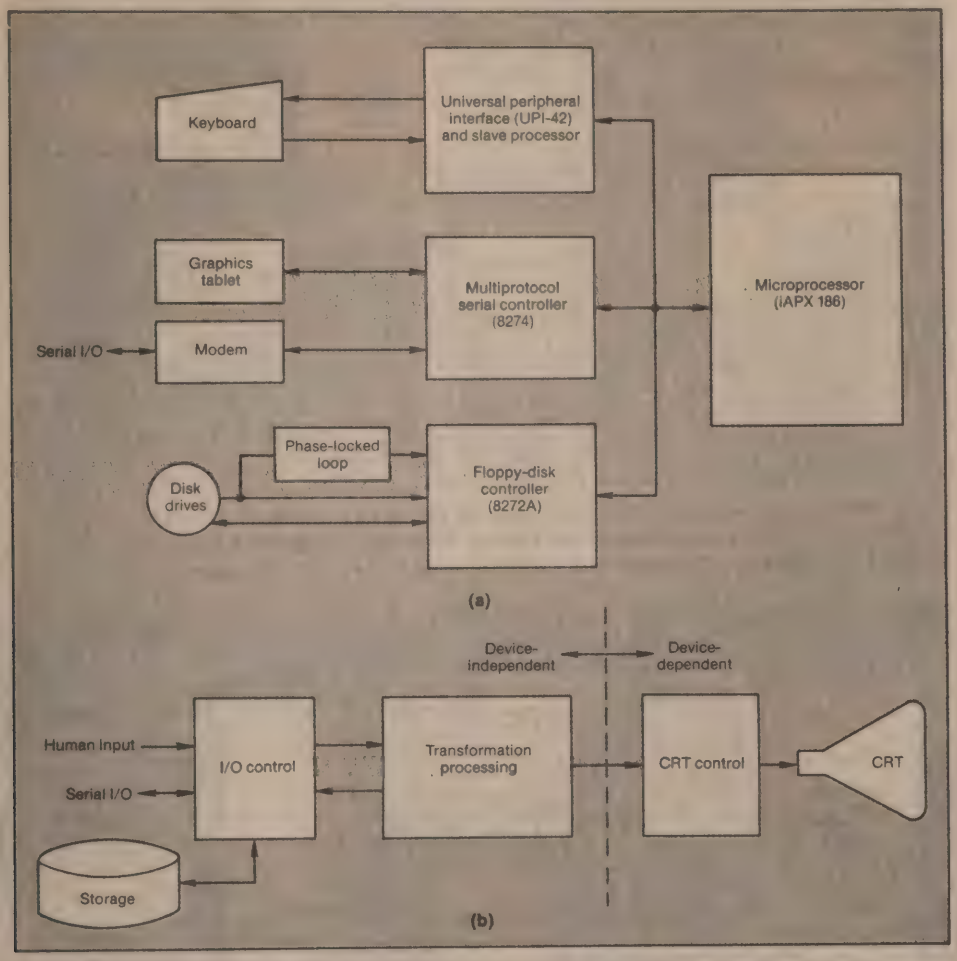
Reprinted from **ELECTRONIC DESIGN- January 20, 1983**
Copyright 1983 Hayden Publishing Co., Inc.

Order Number: 231310-001

this computer must also be able to run on the work station.

Two of the most basic functions of a graphics system are generating and refreshing images on the CRT screen. Information pertaining to the images is stored in the bit-map memory, where monochrome pixels are represented by single bits and color pixels by groups of bits. Lines and arcs defined in normalized screen coordinates must be converted into images of the physical object.

In a bit-mapped raster graphics system, lines described by a transformed display list are reduced to a series of dots and placed in the image memory. The selection of the dots that will be activated is achieved through a scanning conversion algorithm, which must create lines that appear very smooth, start and end as expected, and look symmetrical no matter in which direction they are drawn. The algorithm is repeated thousands of times to draw a single picture and thus must operate as quickly as possible. At the same time, the image in memory must be repainted on the screen 30 times/s for

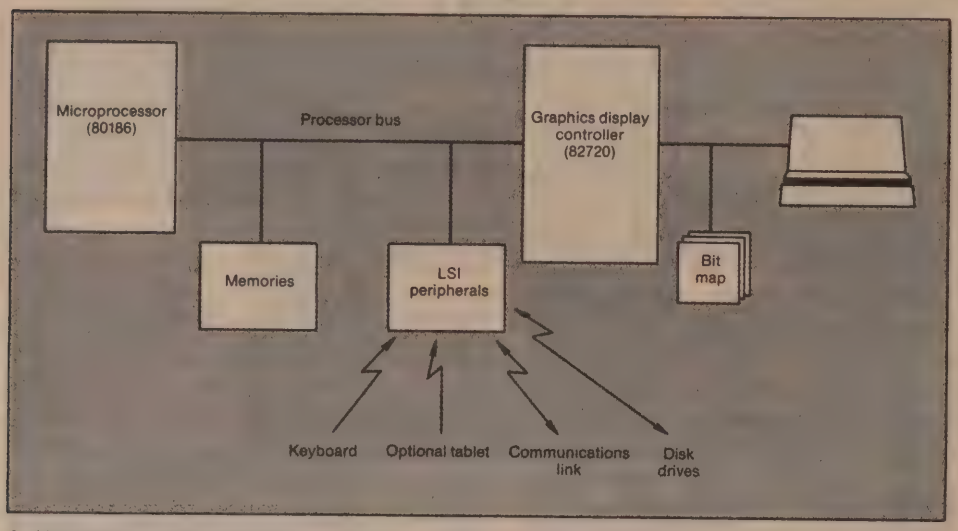


1. A graphics I/O subsystem for an intelligent work station consists of input peripherals (a keyboard and tablet), a serial communications link, and mass storage (floppy disks). Intelligence is provided by the microprocessor and the peripheral and memory controllers (a). The three basic tasks performed—I/O, transformation processing, and CRT control—all require data in the form of display lists stored in a data base (b).

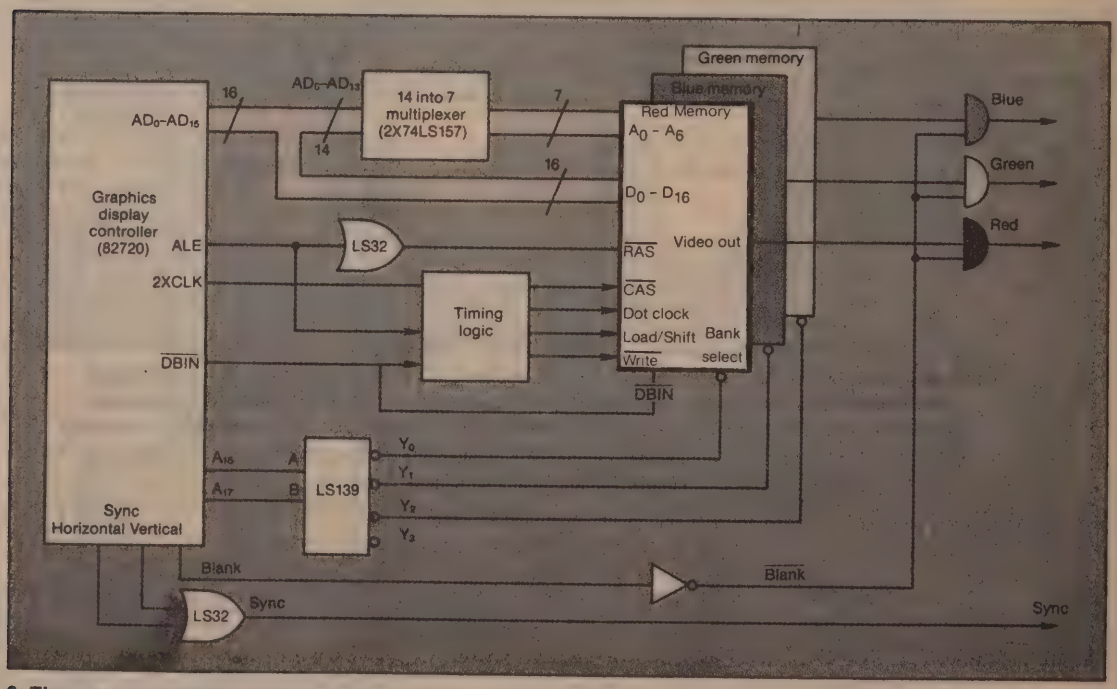
interlaced frames and 60 times/s for noninterlaced frames. Simple tasks, they nevertheless demand a high memory bandwidth.

Unlike other system control tasks, generating graphics figures requires both bit-manipulation and mathematics capabilities. Integer addition and multiplication operations calculate the coordinates of points on a line or a circle. But since pixels generally are neither complete words nor bytes, logical operations must be performed on the bits within the word that contains the selected pixel.

The inner loop of a so-called Bresenham line-drawing algorithm requires two or three addition operations, two comparisons or tests, and the masking of the correct value into the word for each pixel. Algorithms for drawing circles or filling areas are even more complex. In the inner loop of a filling algorithm, for example, the old word must be read from the bit map to determine whether some, all, or none of the pixels are within the area to be filled. If they are, the algorithm tests whether the pixels must be modified and then returns the word to the



2. The 82720 graphics display controller separates the tasks of graphics generation and CRT refreshing from other system tasks. That permits much greater system bandwidth, leading to graphics work stations that not only draw sharp pictures, but also offer color.



3. Three memory planes are implemented in the interface between the bit map and the graphics display controller. Three primary colors—red, green, and blue—are provided, with the controller's upper address bits responsible for selecting the memory planes during read/modify/write cycles.

bit map. Because such algorithms are heavily exercised, they must execute at extremely high speeds to avoid an adverse impact on the system's overall efficiency.

Memory bandwidth is the most precious commodity in a graphics system. In this application, screen refreshing requires that 750,000 bits be read 60 times/s, equating to a bandwidth of almost 6 Mbytes/s. The picture refreshing, therefore, has the highest-priority access to memory because any missed readings show up as noise in the picture, a situation that sometimes occurs with simple systems possessing a single-microprocessor, single-memory scheme.

In the latter type of design, one processor handles all functions except refreshing, which is implemented by a discrete counter arrangement or a simple CRT controller chip. Nevertheless, the refresh memory bandwidth always slows down the microprocessor. That loss of speed can be eliminated simply by separating the processor's memory system from the bit map, a process that effectively doubles system memory bandwidth.

The 82720 graphics display controller can provide the means of separating graphics generation and CRT refreshing from the other tasks and also perform the two tasks quickly and concurrently with the others. Residing between the microprocessor and the bit-map memory and video logic, the controller refreshes the CRT like other CRT controllers, converts high-level commands into images by placing the proper data into the correct bit

A look into graphics fundamentals

The graphics data found in graphics display lists typically describes objects in the real-world Cartesian coordinate system conforming to the axes X, Y, and Z (see the figure). Graphics data does not take the form of one bit for every point on a line; rather it represents higher-level forms such as the end points of a line and

```
WALL: OBJECT
      MOVE TO [X, Y, Z]
      DRAW LINE TO [X2, Y2, Z2]
      DRAW LINE TO [X3, Y3, Z3]
      DRAW LINE TO [X4, Y4, Z4]
      DRAW LINE TO [X1, Y1, Z1]
END WALL
ROOF: OBJECT
      •
      •
      •
HOUSE: OBJECT
      WALL AT [T1]
      WALL AT [T2]
      WALL AT [T3]
      WALL AT [T4]
      ROOF AT [T5]
      WINDOW AT [T6]
      WINDOW AT [T7]
      DOOR AT [T8]
END HOUSE
```

the starting, ending, and center points of an arc.

The coordinate system handles physical measurement units such as inches, feet, or meters, which are typically represented in a computer by 16- or 32-bit integers or by floating-point formats. Ultimately, complex graphics structures are stored in a data base in a hierarchical form consisting of lists of X, Y, and Z coordinates.

The first step in designing a CRT subsystem involves selecting the resolution and scanning rates. All conventional raster-scanning monitors have a display area that is wider than it is high in the ratio of 4 to 3 (called the aspect ratio). For pixels to be square—equally spaced in both the X and the Y direction—the number of horizontal pixels must be 4/3 the number of vertical pixels. This is expressed as 4H-3V, where H and V represent the number of horizontal and vertical pixels respectively. Resolution depends on the total quantity of pixels, which must be a power of two. If it is not, the number of pixels must be rounded to the next highest power of two, in which case some bits will be wasted. Furthermore, the number of horizontal pixels must be organized as an even number of 16-bit words.

To prevent wasted bits, the number of vertical and horizontal pixels are chosen as large as possible without exceeding a power of two. For the display in question, 512H by 512V = 2^{18} = 262,144 pixels. A screen format of 576H by 432V normally meets all requirements. The total number of pixels is then 248,832, and the ratio of horizontal to vertical pixels (576/432) is correct. Furthermore, the number of horizontal pixels makes exactly 36 16-bit words.

After figuring the aspect ratio, the format of the bit-map memory is the next item to be considered. The screen contains about 250,000 pixels, each of which can be either black or one of seven colors. These eight shades can be represented by three bits/pixel ($2^3 = 8$), meaning that the bit-map memory must handle about 750,000 bits. The organization of the memory, however, must be determined according to the various tradeoffs.

The entire memory must be accessed 60 times/s since that is the rate at which the image must be painted to prevent flickering. That equates to a refresh rate of 16.7 ms. As a rule of thumb, the monitor displays information 75% of the time and is blanked for retracing operations 25% of the time. Thus the whole memory must be read and sent to the CRT during a 12.5-ms interval (16.7 × 0.75), which constitutes the active

map, and interfaces easily and simply with proprietary microprocessors.

The 82720 accepts high-level commands (such as DRAW LINE, DRAW ARC, and FILL RECTANGLE) and executes them at much faster speeds than general-purpose microprocessors, primarily because it is a dedicated graphics hardware processor. Burst drawing rates as high as 1 pixel every 800 ns can be achieved. Screen refreshing is handled directly by the controller. The displayed portion of the bit-map memory can be configured to allow the display to be scrolled through memory in any direction. The horizontal and sync periods both are fully programmable, as is the position of the sync pulse in the blanking interval. Furthermore, the controller can be programmed to refresh low-cost dynamic RAMs. In the design being considered, the 82720 offloads the microprocessor from low-level graphics tasks, as shown in Fig. 2.

For the bit-map interface, the memory is implemented as three planes, each 16 kwords by 16 bits, with each plane driving red, green, or blue (Fig. 3). The upper address bits—A_{16} and A_{17}—select the memory planes during read/modify/write cycles but are ignored during screen refreshing cycles.

The graphics display controller generates the Row Address Strobe (RAS) signal for the dynamic RAMs, but the remaining timing signals must be supplied by external devices. These signals are produced by a state-machine timing generator consisting of a 4-bit counter and two flip-flops. The state machine synchronizes itself with RAS after

portion of a frame.

To meet these requirements, it is helpful to break the bit map into three planes of 432 by 576 bits. While the screen is being refreshed, data is read from the same address in each of the three planes and sent serially to the screen. The memories can then be arranged as three 16-kword-by-16-bit arrays, requiring a memory cycle time of 800 ns and consequently permitting the use of relatively slow, low-cost 16-kbit dynamic RAM chips.

When drawing graphics figures, memory can be treated as a single large plane divided into three primary colors: red, green, and blue. Thus the low-order memory could represent the color red; the middle-order memory, green; and the high-order memory, blue. Each primary color requires the setting of just 1 bit/pixel. However, a secondary color—cyan, yellow, or magenta—necessitates setting 2 bits/pixel. Therefore, drawing in a secondary color takes two memory cycles/pixel and is slower than drawing primary colors. If this creates system problems, additional hardware can be used to draw more than one plane at a time. However, in the system example, drawing speeds are not only met, but also exceeded without relying on extra hardware.

Starting with the vertical refresh rate of 16.67 ms/frame, the basic timing can be analyzed. From the 16.67-ms figure, subtract the 1.25 ms required by the monitor for its vertical blanking. That leaves 15.42 ms for scanning the 432 lines on the active portion of the display. Dividing 15.42 ms by 432 lines gives 33.5 μs/line, equivalent to a horizontal scan rate of 28 kHz.

Vertical retracing requires 7 μs/line, and the active portion of each line is 35.7 μs. Subtracting 7 μs from 35.7 μs leaves 28.7 μs. During this time 576 pixels are displayed, for a pixel period of 28.7 μs/576 or 49.8 ns. This corresponds to a dot clock rate of 20.07 MHz, which is chosen as the system's basic clock rate.

Display lists and commands pass from the I/O subsystem to a unit that executes the transformation tasks. Transformations are primarily mathematical operations performed on the display units. Depending on the command, this module edits display lists, organizes them for display on the screen, or edits the display-list data base. By editing a display list, objects in the physical coordinate system can be created, destroyed, moved, or changed. Transforming a display list into a form compatible with the display is necessary, as the data base can have an unlimited real-world coordinate system in three dimensions, but the CRT screen is limited to only two dimensions.

Transformation tasks place a heavy burden on a microprocessor. For instance, in a typical transformation, a matrix multiplication is performed for every point on an object's display list. In three dimensions, each point requires multiplying a 4-element vector by a 4-by-4 matrix. Some of the elements are always zero, but the operation still takes 13 multiplications and 9 additions. A two-dimensional display list requires 4 additions and 4 multiplications. A typical display can contain hundreds or thousands of lines, each of which has two end points. Therefore the speed of matrix multiplication significantly influences system performance.

The coordinate system supported by the design example is three-dimensional and employs 32-bit integers. The system CPU executes 32-bit integer matrix multiplications at high speed. In conjunction with the graphic display controller, the drawing task is offloaded from the CPU, which in turn maximizes the central-processor time that can be allocated for those matrix multiplications. Waiting time is now much lower than in conventional systems. However, if a system requires floating-point transformations, the best high-speed performance is achieved with the addition of a numerical coprocessor.

the 82720 has been initialized. Figure 4 shows the complete schematic for each plane of the bit-map interface.

The remainder of the hardware design interfaces the graphics display processor, the processor memory, and the other peripherals with the 80186 microprocessor. The task is simplified by the processor's on-board chip-selection logic and wait-state generators. Furthermore, because of the processor's highly integrated architecture, the size of the overall hardware is quite small.

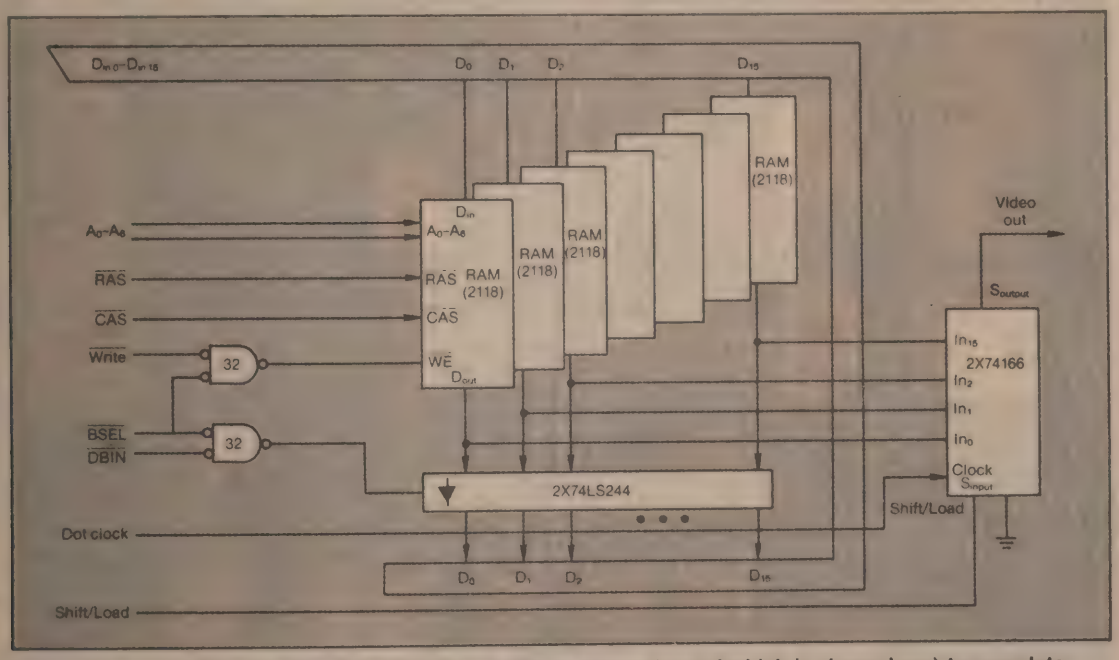
Joining processor and controller

Connecting the graphics display controller to the microprocessor is a simple task, as the processor's Data, Read, and Write signals are completely compatible with those of the 82720. However, because the controller has no chip-selection input, the Read or Write signals must be qualified through external hardware.

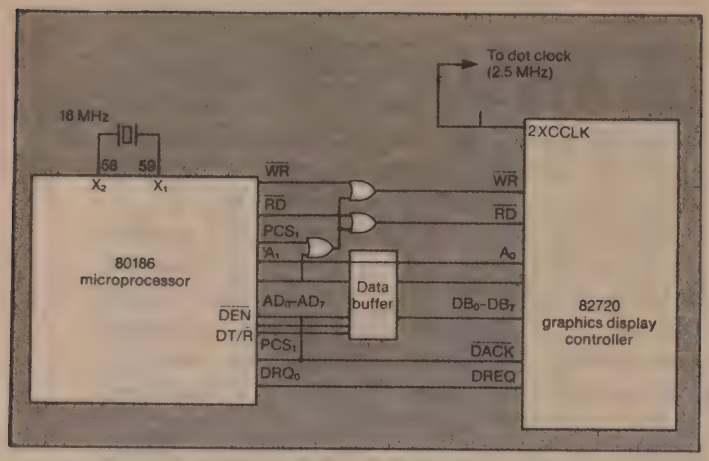
A number of chip-selection lines on the microprocessor can be programmed to place peripherals either in memory or in the processor's I/O space. Two gates are added to qualify the Read and Write signals. The DMA channel on the 80186 uses a second chip-select input as the Acknowledge signal, and data buffers are used to prevent bus contention at the end of a processor read cycle (Fig. 5).

Without buffers, the display controller must remove its data from the multiplexed address and data lines before the processor puts out the next address. At an 8-MHz clock rate, the processor requires that peripherals and memory vacate the bus in less than 85 ns; however, the standard speed of the controller is 100 ns. A faster version, the 82720-1, can be used, but it requires faster memory chips. A more cost-effective solution is simply adding buffers, if board space permits.

Serial communications to both the host and the optional tablet are handled by a multiprotocol serial controller (the 8274), which takes care of the host's synchronous and the tablet's asynchronous



4. The bit-map memory interface contains three address planes (one of which is shown here) to complete the graphics system. The RAS signal for the RAMs is generated by the graphics display controller.

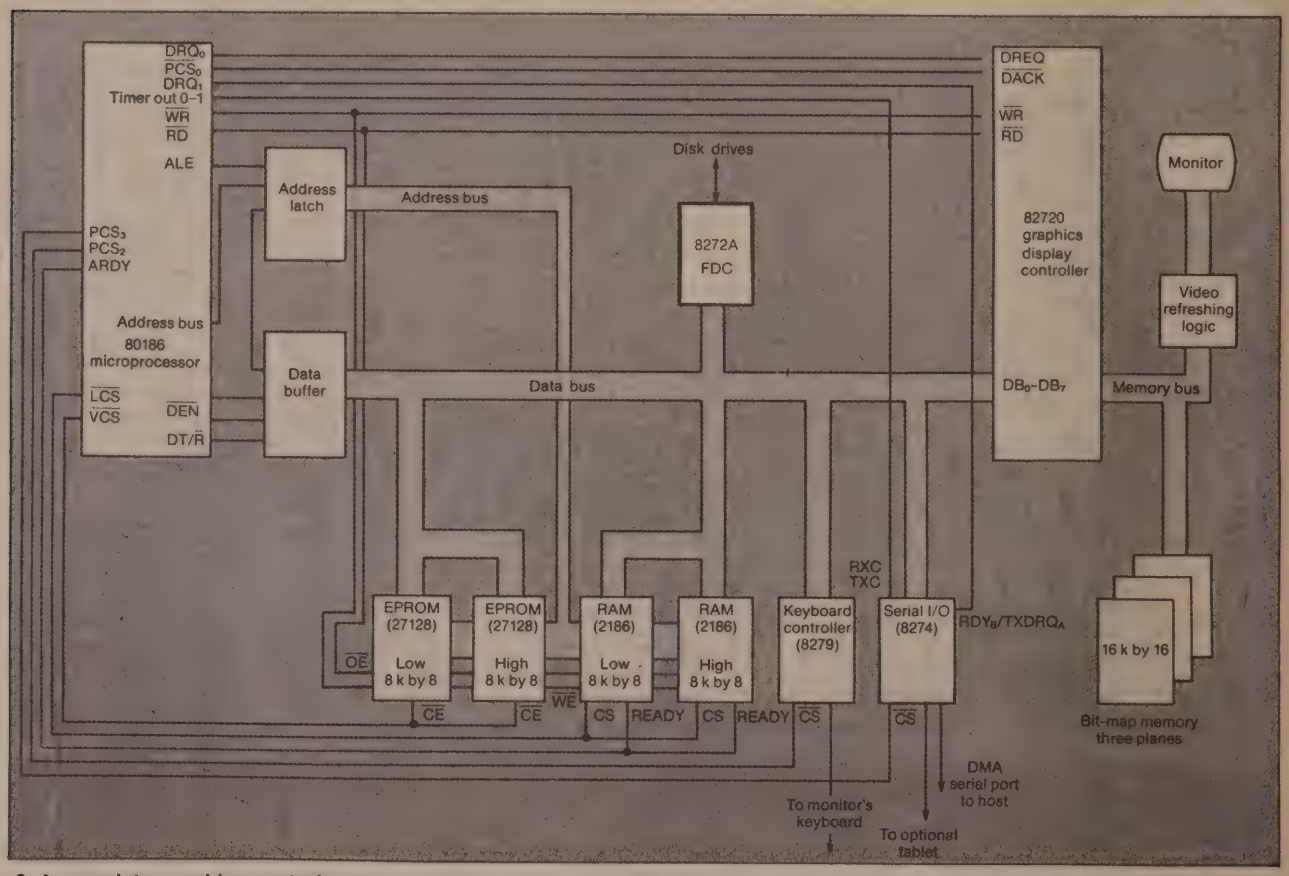


5. The interface between the 82720 and the system microprocessor is simple to implement because all of the processor's signals are compatible with the controller. It is necessary, however, to use external gates to qualify the $\overline{RD}$ or WR signals.

requirements. Interfacing is accomplished simply by connecting the buffered data bus, the latched lower-address lines, the Read and Write signals, and the chip-select. A final link brings the microprocessor's counter-timer output into the multiprotocol serial controller as a baud-rate clock. No buffering of the TTL support circuitry is necessary.

Universal chip interfaces keyboard

A universal peripheral interface chip (the UPI-42) serves as the keyboard interface and is programmed to scan the keyboard and interrupt the processor only on detection of a valid debounced keystroke. Mass-storage subsystems are managed by the 8272A floppy-disk controller. An external phase-locked loop circuit generates all of the timing signals reequired to connect a 5¼-in. drive to the system. On the microprocessor side, a DMA channel provides the link to the floppy-disk controller. Thus



6. A complete graphics control system is centered around an 80186 microprocessor and the 82720 controller. Local storage is provided by 32 kbytes of EPROM and 16 kbytes of RAM. The system comprises 85 chips and is housed on a single 12-by-12-in. printed-circuit board.

the processor has a high-speed disk interface, which loads it lightly.

To complete the graphics system illustrated in Fig. 6, 32 kbytes of EPROM and 16 kbytes of RAM support the microprocessor's program and display lists. The two EPROMs (27128s) come in 28-pin packages, thereby saving board space.

Hooking up the RAM chips is almost as straightforward. Since the microprocessor is a fully byte-addressable device, it can write bytes as well as words to the RAM. The chip-select input for the low (even) address RAM must be qualified with address A_0 at a logic zero, and the high (odd) address RAM must be qualified by the processor's Byte High Enable signal (BHE). The RAMs, designated 2186, have built-in controllers.

Since dynamic RAMs latch addresses on the leading edge of the chip-select signal, they must be qualified with the processor's Address Latch Enable signal to ensure that selection is made only after the address is valid. Then, a RAM latches the data to be written on the leading edge of the write pulse. The microprocessor's write signal must be delayed by one-half of a clock cycle to guarantee that data is valid at the correct time.

At this point, the design meets all of its performance goals. The system draws lines and circles at about 120,000 bits/s. That is approximately 82,000 pixels for a display consisting of even amounts of the three primary colors, as well as three secondary colors, and white. The 500 vectors of 25 pixels each can be drawn in about 0.15 s, six times faster than the 1-s requirement. The worst case—drawing all lines in white—can be accomplished in about one-third of a second. These specifications are satisfied when the graphics display controller is running from a 2.5-MHz clock. Drawing is performed only during retracing and the 82720 is programmed to use three memory cycles of each horizontal retrace for memory refreshing.

All of the components fit on a board measuring 12 by 12 in., so that the desktop size requirements are satisfied. The electronic components occupy about 100 in.^{2} of the low-cost, double-sided printed-circuit board. □

Bibliography:
Bresenham, J.E., "Algorithm for Computer Control of a Digital Plotter," IBM System Journal, 1965, 4(1) pp. 25-30.

Graphics Chip Makes Low-Cost, High Resolution Color Displays Possible

by Mark Olson and Brad May

The making of displays that are both high-resolution and low-cost is the key to producing equipment for both the automated office and the engineering workstation. Through the introduction of 16-bit μPs such as Intel's iAPX 8088, 80186 and 80286, the processing power has been made available to perform very sophisticated functions for the user while making the human interface very simple.

That processing power can be unnecessarily drained, however, if the μP is burdened with the entire task of graphics display. Such a burden can fill up a significant part of the processor's I/O bandwidth, slow down the refresh rate of the display, and decrease the computational power of the CPU.

Intelligent peripheral ICs offload processing tasks from the CPU.

The logical way to avoid such limitations is to dedicate a specialized processor to the handling of display function. It should be capable of accepting high-level commands to minimize the burden on the CPU, as well as optimizing the execution of such commands through raster operations implemented in hardware at the device level.

Such a chip is Intel's 82720 Graphics Display Controller (GDC). It has features that give systems a fast drawing speed while reducing graphics display costs by 60% or more. It achieves these results by taking over the drawing and refresh functions from the CPU, by allowing the use of dynamic RAM's instead of static RAM's, and by reducing the overall parts count needed to create a complete graphics system.

The implementation of the drawing task is a major feature of the GDC. Other graphics chips perform only the display refresh function, leaving the more complicated drawing function entirely to the CPU. Since the CPU is doing every pixel of the drawing function on these systems, they also require faster bit map RAM than with the GDC. The GDC, on the other hand, is capable of handling the drawing function itself, drawing such objects as characters, slanted characters, points, lines, arcs, rectangles, and slanted rectangles based only upon lengths, slopes, and arc centers supplied by the CPU. The GDC's processing, moreover, takes place concurrently with the processing of the CPU.

2048 × 2048 Resolution

With its 4 Megapixel addressability, one GDC can handle a monochrome display with resolution as high as 2048 × 2048, and multiple GDC's can be linked to provide even higher resolution, such as color displays at 2048 × 2048. The chances are, however, that the GDC's full power will not be used in most applications. The typical

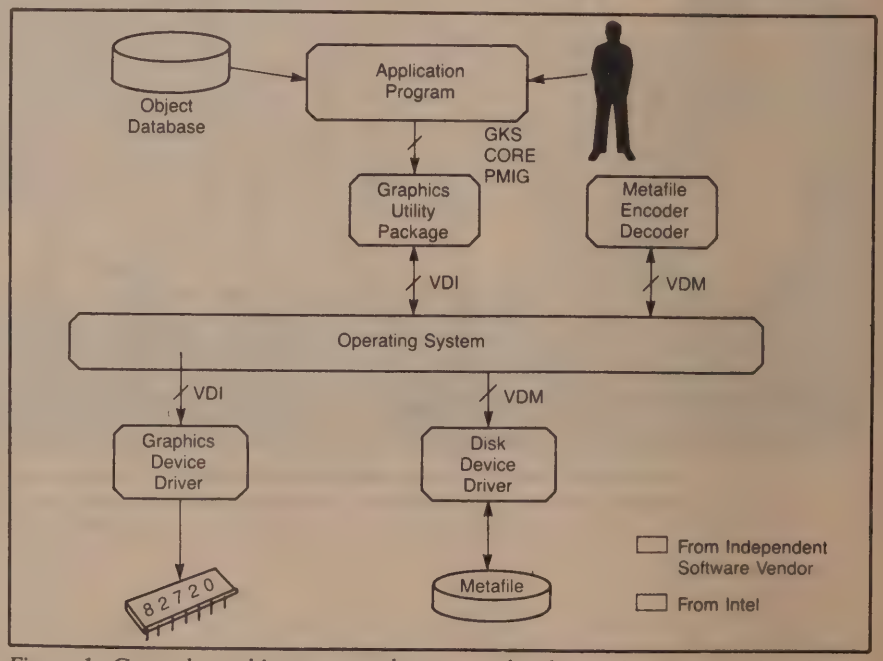


Figure 1: General graphics commands are translated into the VDI interface level and then into driver device commands.

Mark Olson and Brad May are Product Marketing Engineers for Peripheral Components Operation, Intel Corp., Santa Clara, CA 95051.

Reprinted from **DIGITAL DESIGN** © April 1983, Morgan-Grampian Publishing Company, Boston, MA 02215
231315-001

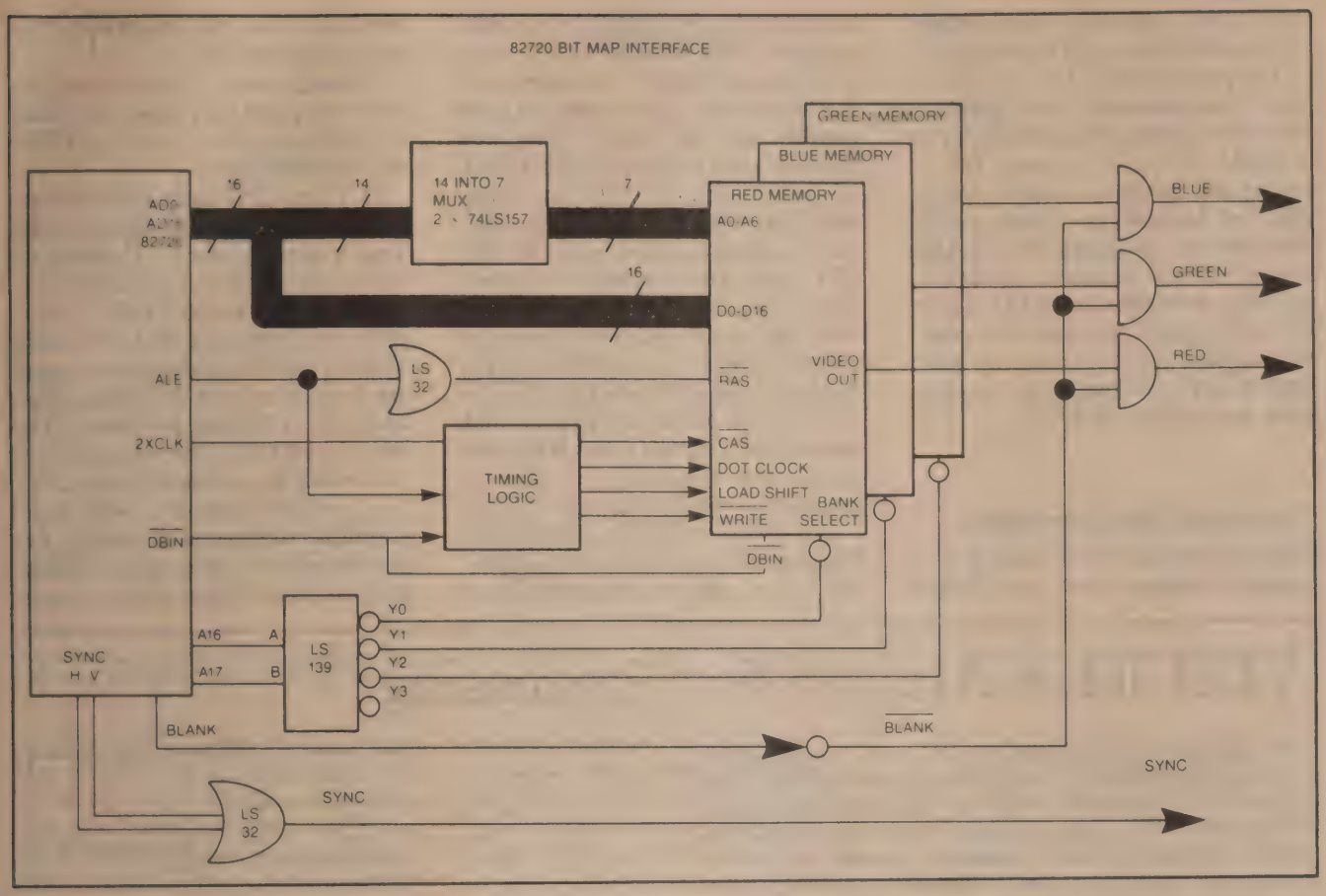


Figure 2: The memory is broken up into three planes, with each plane feeding one of the primary color guns of the CRT.

product considered high resolution for office automation applications is a 512 × 512 pixel monochrome or color display.

These latter restrictions are not imposed by the GDC, but rather have more to do with the cost of display monitors, the amount of RAM memory needed to support such displays, and the adequacy of such displays for most applications. It is possible to build "super graphics" boards with a GDC, such as the 1K by 1K pixel by 8 color plane graphics display designed by Phoenix Computer Graphics (Lafayette, LA). Such a display is capable of rendering 256 different colors on a high resolution screen.

Even higher performance can be achieved through the use of multiple GDC's to support multiple display windows, increased drawing speed, or increased bits per pixel. For multiple display windows, each GDC can be used to control one window of the display. For increased drawing speed, multiple GDC's can be operated in parallel. For increased bits/pixel, each GDC can contribute a portion of the number of bits necessary for a pixel.

Although the GDC is intended primarily for raster-scan graphics, it can also be used as a character display controller. It is capable of supporting up to four screens of data containing 25 rows by 80 columns, or one screen containing up to 100 rows by 256 characters.

Office Automation Display

High performance applications can stretch the usage of the GDC from low-end to high-end engineering displays, but research has shown that for office automation products, a 512 × 512 pixel display is quite acceptable, and that color is often a requirement. These requirements mesh with a major factor in display—the cost of the CRT. In OEM quantities, for example, one could expect to find a 512 × 512 monochrome display for under $100, a 256 × 256 color display (TV quality) for about $150, a 512 × 512 color CRT in the $300 range, and a 1K × 1K color display in the $800–$1000 category.

To give an example of the type of display that can be built for new office products using the GDC, consider a 512 × 512 pixel by 3 color plane combination CPU and graphics display on a single 12″ by 12″ board. Such a display is capable of generating 8 colors.

The list of parts (**Table 2**) comes to about $175 for 85 IC's taking up 104 square inches of board space. Even that parts count could be reduced by replacing the 48 16K DRAMs with 12 64K DRAMs—if a 4K × 16 bit DRAM were available. A very important note about the parts list is that the design is implemented with inexpensive 2118 dynamic RAMs. The design does

not require the faster, more expensive, and less dense static RAMs.

The parts count is low enough so that the processor and graphics controller can be placed together in a single 12" by 12" board. This is important because small overall size and footpad are selling points for desktop workstations. System speed is also enhanced when the graphics controller and CPU are on the same board, because their communication need not take up bus, inter-board bandwidth or experience any additional delays.

Pipelining Transformations

More important than putting the graphics display on the same board as the CPU is the level of communication between the CPU and graphics controller. If the burden of transformation processing is left entirely to the CPU while the graphics chip is used only as a CRT controller, then the CPU must communicate one bit per pixel to update a display. With the GDC, the CPU input takes higher level forms such as the slope and length of a line, the length and center point of an arc, or the key coordinates of a rectangle. Since the average line on a screen is about 25 pixels, that means that 25 times fewer CPU bus cycles are required to draw a graphical object with the GDC. These CPU cycles (an average of 50 μs each to calculate the graphical object and communicate it to the GDC) are the determining factor in drawing rate.

Viewed from a larger perspective, there are four tasks that must be performed by a CPU-graphics chip combination:

(1.) The CPU must calculate the higher-level graphics operations. This is done by the CPU and it involves the processing of macro-operations such as the CORE, GKS, PMIG or other graphics protocols. These general graphics commands are translated into an intermediate level, the VDI interface level (**Figure 1**) and then into device driver commands by software in the CPU.

(2.) Then, these lower-level graphical objects such as the key parameters for lines, arcs, characters, and rectangles, must be trans-

VLSI Takes Aim At Text Processing

The concept of co-processing is not a new one. Intended as a way of offloading computationally intensive tasks from a host CPU, it has been around at Intel since the introduction of the 8087 numerics processor and the 8089 I/O machine. A more recently developed product, the 82720 Graphics Display Controller is designed to bolster system performance by offloading graphics control chores from the CPU. The chip accepts high level commands from the CPU and, using its own drawing processor, accesses the required positions in the bit-map and handles the processing and display control functions.

Building on the success of these parts come two new co-processors designed to partition system intelligence even further. The 82586 is a communications co-processor designed to bridge the characteristics of CPU and network data rates. Its FIFO buffer and DMA facilities make it possible for a CPU to operate at the full Ethernet 10 Mbits/s transfer rate even in the face of continuous bursts of network data traffic.

Intel's most recent introduction is the 82730 text co-processor. Printers and other hard copy peripherals have supported additional text processing features such as proportional spacing and simultaneous superscript and subscript for some time. Implementing these features on the display screen has traditionally been a costly procedure. Thus, it is typically not done and screen displays often are not identical to their hard-copy printouts. Aimed to solve this designers headache, the 82730 has its own DMA capability and communicates asynchronously with the CPU via shared memory messages. It supports the generation of high quality text displays through features like proportional spacing, simultaneous superscript/subscript, dynamically reloadable fonts and user programmable field and character attributes. In addition, when coupled with the 82720 Graphics Display Controller (**Figure 1**) the 82730 provides flexible mixing of text and graphics simultaneously on the same display.

—*Wilson*

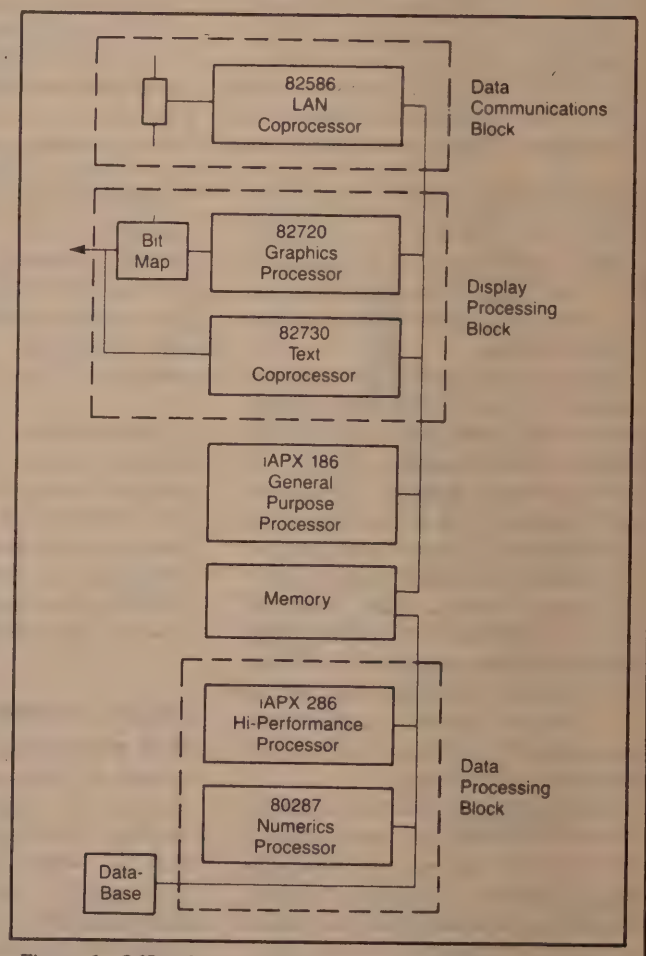


Figure 1: Offloading system tasks is simplified by new VLSI devices.

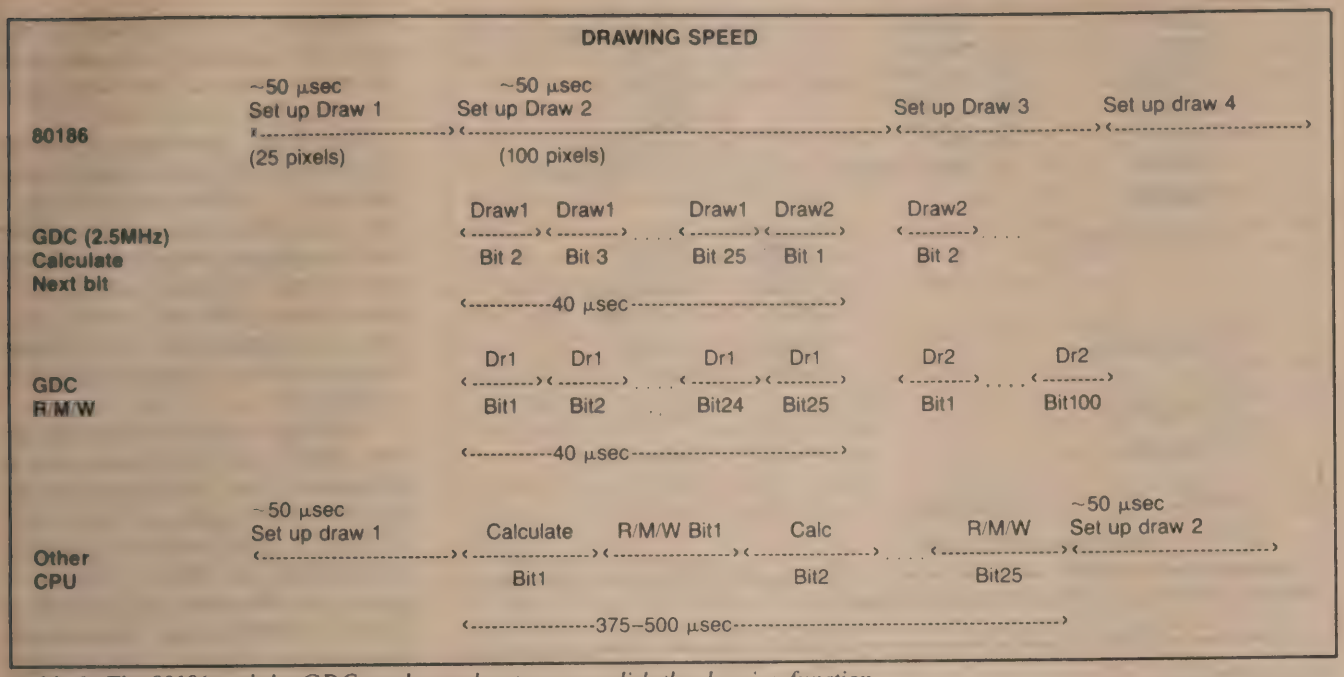


Table 1: The 80186 and the GDC work together to accomplish the drawing function.

formed into changes in the actual bits. This function is performed in hardware in the GDC concurrently with any level one processing done by the CPU. Other graphics controllers leave this task to the CPU to execute in software. The contrast is that, in such systems, the CPU must resolve the graphical object down to every point on a line, while with the GDC it need only designate the endpoints.

(3.) With the actual bits for the bit map calculated, they must be placed in the bit map memory. This involves a read-modify-write operation that requires three CPU cycles using other methods. With the GDC these operations are not the responsibility of the CPU. The GDC pipelines its execution so that it is calculating the next bit to change while it is executing the read-modify-write cycles.

(4.) Finally, the bit map memory must be dumped into the CRT. This is the refresh function performed by other graphics chips as well as the GDC.

The summation is that other systems require the CPU to process steps one to three serially, leaving only step four for the graphics controller. Systems with the GDC require the CPU to process only step one, with the GDC concurrently processing steps two through four. The GDC has another advantage in that during the transformation process in step three, the GDC executes the algorithms in hardware while a CPU must execute the algorithms in software. The algorithms are exactly the same in both cases. They are the Bresenam algorithms from IBM, in which the next pixel to be drawn becomes a binary decision between two pixels.

The execution of these algorithms is a crucial drawing time factor, because they are invoked many times for each updated screen. Consider that, in the inner loop of Bresenam's "line drawing algorithm," there are two or three additions, two comparisons or tests, and the masking of the proper value into the word for each pixel. The algorithms for drawing circles or filling areas are even more complex. In the inner loop of a fill algorithm, the old word must be read from the bit map, then tested to see if all, some, or none of the pixels are within the area to be filled. Next, it tests whether some or all of the pixels must be modified. Finally, the word must be returned to the bit map.

These algorithms are heavily used and the speed with which they can be executed has a direct effect upon the overall system efficiency. If they must be executed by a µP, the instruction fetching process slows down the calculations to a drawing rate of 15–20 µs per pixel. With a hardware implementation of these algorithms in the GDC, the calculations can be speeded up to achieve a drawing rate of 1600 ns (2.5 MHz version) or 800 ns (5 MHz version) per pixel.

Methods Of Refresh

In the fourth step, the dumping of bit map memory into the CRT, there are some differences between graphics controller chips. Motorola's MC6845 CRT controller, for example, uses a split-cycle refresh method in which each refresh cycle is alternated with a drawing cycle in which the µP updates the bit map. This gives the MC6845 a 50% drawing bandwidth.

With the GDC there are two drawing modes. The first is a "draw anytime" mode which replaces CRT refresh cycles with drawing cycles. This is the fastest mode, but it does result in on-screen disruptions. The second mode, which does not disrupt the on-screen display, draws only during the vertical and horizontal retracing periods. This gives the GDC about a 25%

1	80186	1	74LS04	1	20 MHz Clock
1	82720	1	74LS73	2	27128
2	74LS157	9	74LS244	2	2186
1	74LS139	8	74LS166	1	8274
1	74LS161	3	74LS32	1	8042
1	74LS11	2	8286	3	Connectors
1	74LS00	1	8 MHz Crystal	1	12×12 2 Layer PC

SUMMARY:

4 VLSI Controllers	80186	Processor
	82720	Graphics
	8274	Serial Link
	8042	Keyboard
4 VLSI Memory	27128	EPROM
	2186	IRAM
48 16K DRAMs	2118	DRAM
29 MSI/SSI	—	Buffers/Glue

TOTAL: 85 IC'S →104 Sq. Inches

Parts Cost →About $175

Table 2: Parts list for 512×512×4 Color Display.

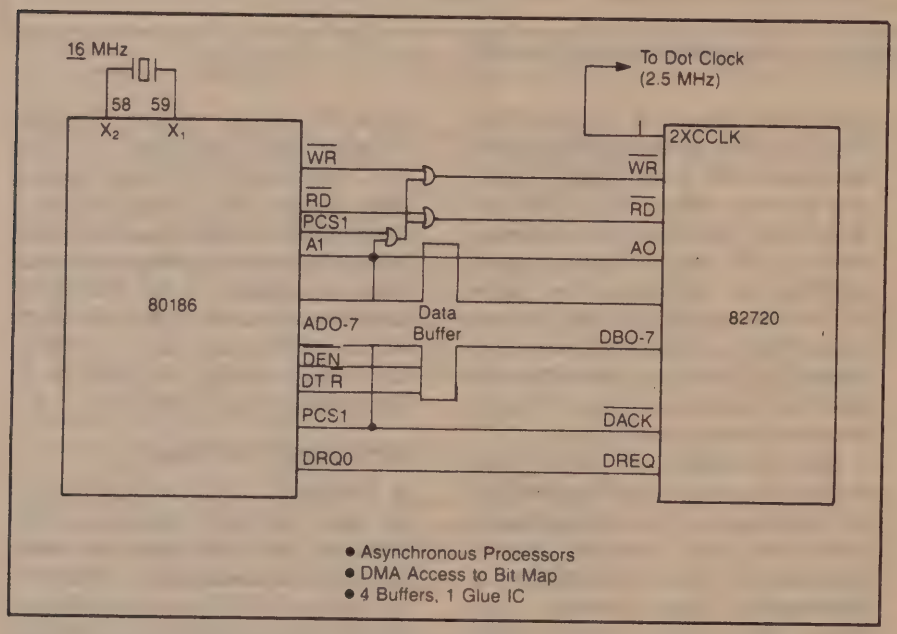


Figure 3: The two chip selects are OR'd together to qualify the R/W signals.

drawing bandwidth. At first glance that gives the GDC a disadvantage in drawing rate, but the fact is, with its pipelining and hardware execution of transformations, the GDC makes much more efficient use of its bandwidth. The critical timing factor is the amount of CPU participation in the drawing process, not the refresh bandwidth of the graphics controller. Another tradeoff is that, with its split-cycle architecture, the MC6845 requires RAM memory that is twice as fast as that required by the GDC in the same application.

Inexpensive RAM Is Fast Enough

Applying this perspective, one can begin to build the display with parts listed in **Table 2**. First one notes that a square display, as indicated by the 512×512 pixel initial specification, is not pleasing to the eye. It is much more appealing to have an aspect ratio of about 4:3, in which the number of pixels horizontally is 4/3 the number vertically. If the resolution is such that the total number of pixels is not a power of two, it will be necessary to round up to the next power of two and waste the extra bits.

The pixel arrangement which best meets this requirement is one with a 432 × 576 pixel format. It also meets the requirement that the number of pixels horizontally be an even number of 16-bit words. With three color bits per pixel (red, blue, and green), the total display memory is then about 500×500×3, or 750k bits.

It makes the most sense to break the memory up into three planes, with each plane feeding one of the primary color guns of the CRT (**Figure 2**). This leads to a memory arrangement of 16K × 16 × 3, using 16K dynamic RAMs with a 1K × 16 architecture. When drawing graphics figures, the memory can be treated as one large plane, split into the three primary colors. Drawing in low-order memory could represent red, middle-order could be used for green, and high-order for blue.

One advantage of this 3D memory is that drawing with a primary color requires setting only one bit per pixel. Drawing with a secondary color such as cyan, yellow, or magenta would take two GDC cycles, and creating white from all three colors would take three GDC cycles. If this were an issue, additional hardware could be used to draw more than one plane at a time. As the results will show, however, the drawing speed requirements can be exceeded without any added hardware.

Calculate The Drawing Rate

To see if the proposed design is practical, one should first calculate the drawing rate to see what the user interface will be like. Then one should check the refresh rate to make sure the design is uninterrupted and without flicker.

The proof of the assumption that CPU participation is the dominating factor lies in the 50 μs average time that it takes the CPU to calculate a graphical object and communicate its key parameters to the GDC. Assume that the graphical object is an average line containing

25 pixels, and that there are about 500 vectors on the average screen display.

The GDC's normal clock rate is 2.5 MHz, giving it a 400 ns period (the maximum clock rate is 5 MHz, with a 200 ns period.) It takes four GDC cycles to execute a read-modify-write on a bit (because two read cycles are required), so that the GDC's normal drawing rate is one pixel per 1600 ns. To draw the 25 pixels involved in the average line, then, would take 25 × 1600 ns, or 40 μs. Since this operation is done concurrently with CPU processing, the GDC will be waiting for the next graphical object by the time the CPU is ready.

If the screen were filled with nothing but 25-pixel vectors, then the drawing rate would be determined by the 50 μs average CPU calculation and transfer cycle, averaging about 2 μs per pixel. If all the vectors were white (worst case), then it would take 1.5 secs of drawing time to update the white screen. Since, in the undisturbed-screen mode, drawing is only done during the 25% of the time that the screen is undergoing horizontal or vertical blanking, this would mean 6 secs between updates.

In reality, however, the screen will not be filled with vectors. It will have an average of 500 vectors, and the color distribution could be presumed to be evenly distributed as one-third primary colors, one-third secondary colors, and one-third white. The 500 vectors will require the drawing of 12.5K pixels in monochrome, or 25K pixels with distributed colors. At a drawing rate of 2 μs per pixel, this takes 50 ms to draw. Drawing only during blanking, the screen would be updated in 200 ms.

Under these conditions, it would not help to use the maximum clock rate GDC (5 MHz), but if in some applications the average vector length is 100 pixels, then the CPU calculation-and-bus cycle (50 μs) would remain the same and the GDC's drawing cycle (1600 ns × 100 = 160 μs) would become a limiting factor. Using the 5 MHz GDC would cut that drawing time down to 800 ns/pixel, or 80 μs/vector. The 500 vector average screen would then contain 100K pixels with distributed colors and could be drawn in 80 ms. Multiplying by four because the drawing is done during blanking (25% of the time), that is 320 ms. That is a screen update in less than one-third second for a "busy" screen.

Calculate The Refresh Rate

These calculations are of little importance if the display flickers due to lack of refresh. This exercise is actually a demonstration of how the basic GDC clock rate was derived. Assume a non-interlaced display that must be refreshed 60 times per second. That gives a screen refresh rate of 16.67 ms, but on a typical CRT some 4.27 ms of that is blanked, leaving 12.4 ms of active display time. The dot sweep period is the 12.4 ms divided by the number of pixels (432 × 576 = 248.8K), or 49.8 ns. The inverse gives a 20.07 MHz dot clock.

Since the GDC dumps 16 bits from the bit map memory into the

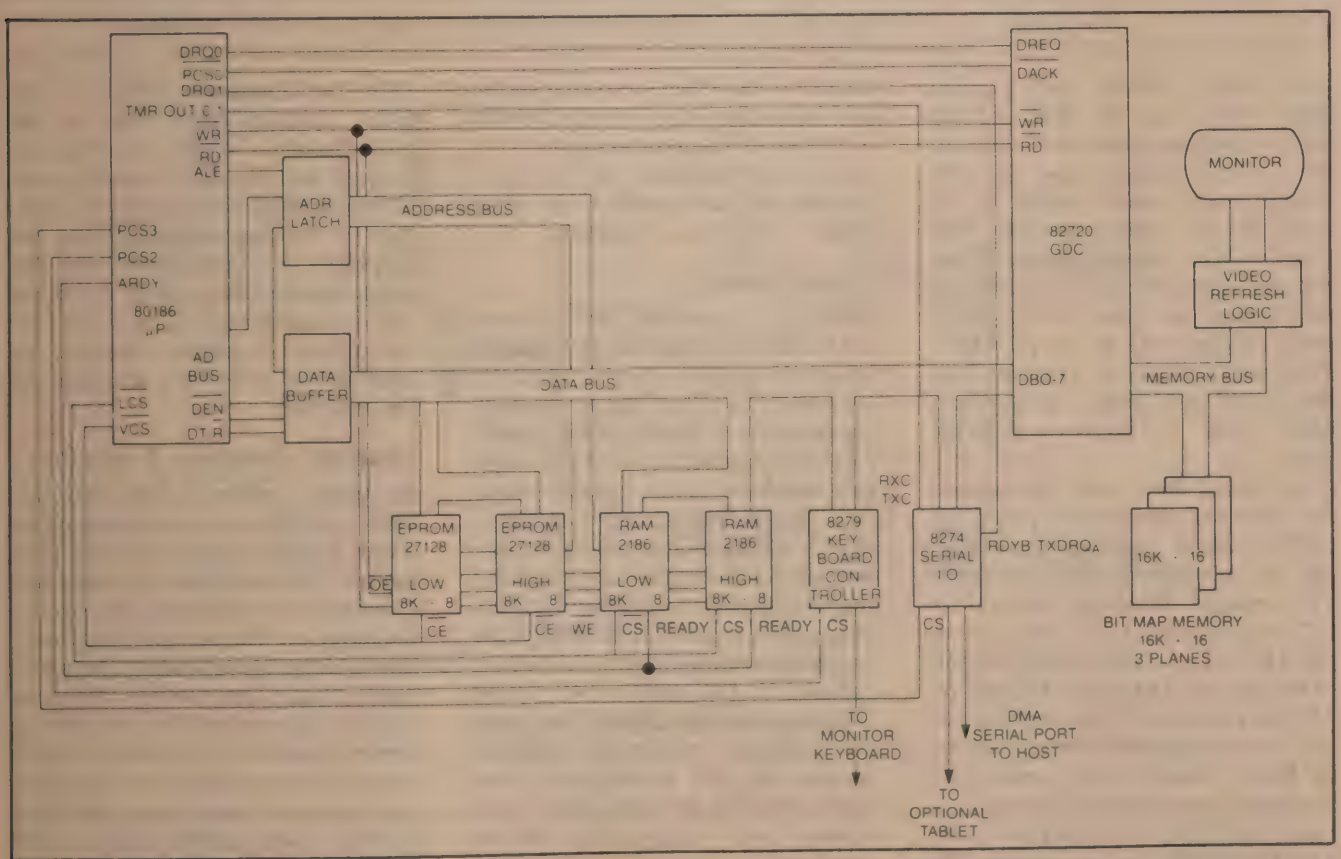


Figure 4: Completed graphics system uses the 80186 and 82720 GDC.

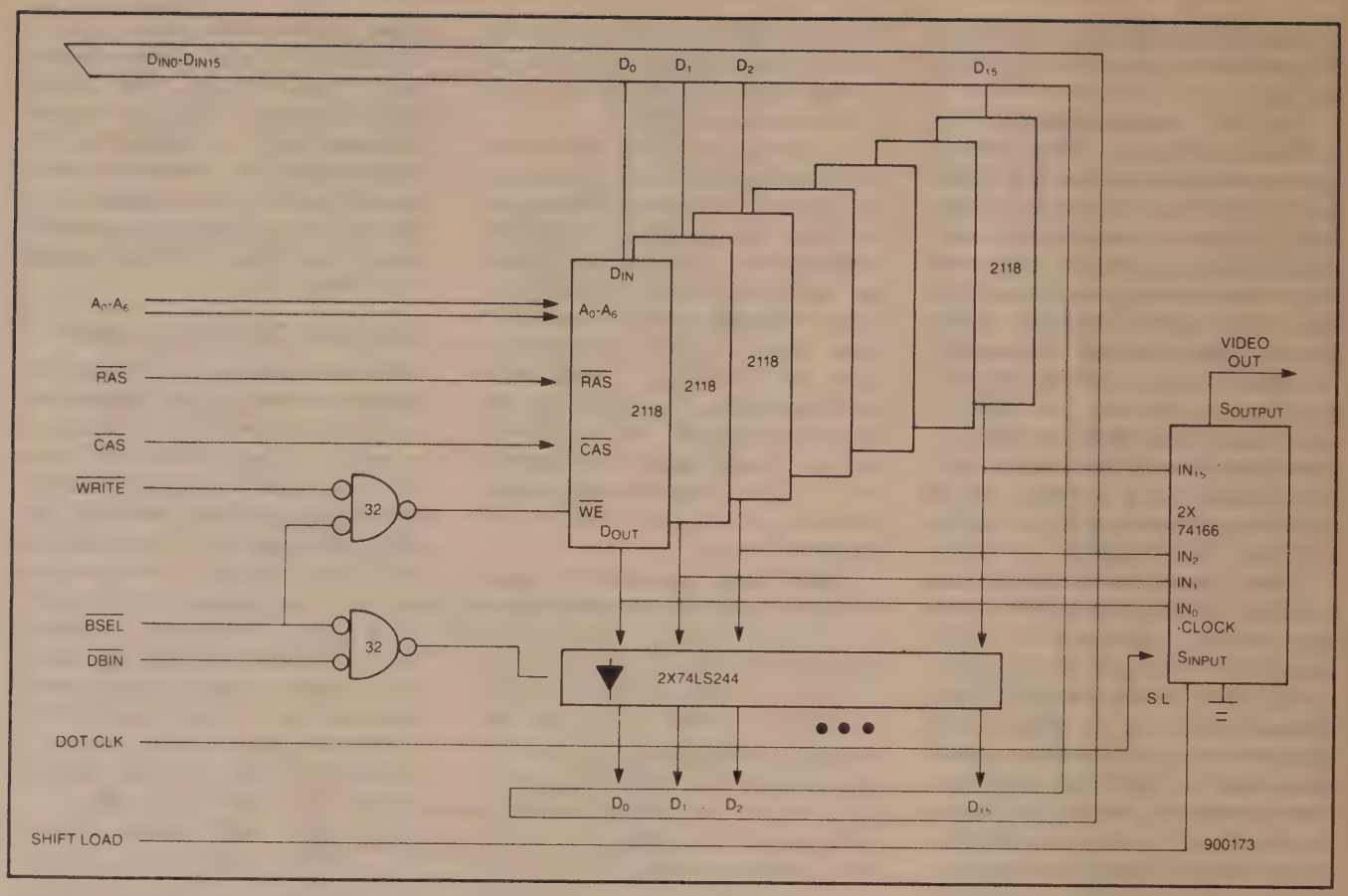


Figure 5: Since the 186 is a fully byte addressable machine, it is possible to write bytes as well as words into the RAMs.

16-bit shift register during each read, and since the shift register then feeds these bits out serially to the CRT, it makes sense that the GDC's read period should be 16 times the dot sweep period. That gives a GDC read period of about 800 ns. With each GDC read taking two cycles, the basic GDC clock period is then 400 ns, or 2.5 MHz. This gives a rock-solid display, and one would only want to go to the 5 MHz GDC to improve drawing rate.

For those who want to examine the blanking intervals to see if the CRT is indeed "typical," the blanking can be further broken down. The vertical blanking interval is 1.25 ms, leaving 15.42 ms to scan the 432 lines on the active portion of the display. Dividing 15.42 ms by 432 lines gives a 35.7 μs period per line, or a horizontal sweep rate of 28 KHz. Time is also needed for horizontal retrace, in this case, 7 μs of horizontal blanking per line. This leaves 28.7 μs to scan the 576 pixels on each line, resulting in the dot sweep period of 49.8 ns. Using a 20 MHz CRT helps keep the costs down, but the GDC can use CRT displays as fast as 80 MHz when higher resolution is required.

Mixed Mode

While it is possible to generate 8×8 characters and slanted characters in the graphics mode, the GDC also offers a mixed mode memory organization to display both characters and graphics drawn from separate windows in the display memory. The advantage of this mode is that it allows characters to be manipulated as 8-bit entities instead of the 64 bits that each would require in graphics mode. Of necessity, the graphics window display memory is reduced in this mode (64K 16-bit words instead of 256K), but even the reduced maximum graphics memory is still a megapixel and quite sufficient for both office automation and engineering display purposes.

In the character window, the GDC operates as it does in the pure character mode, with the exception that the line counter must be implemented externally. In addition to the two windows used for graphics and characters in the mixed mode, two other windows can be supported. These can be designated as either character or graphics windows by a selection on the A17 line.

Panning, Zooming, Light Pen

As special features, the GDC allows both panning and zooming in either graphics, character, or mixed modes. The zoom is accomplished by effectively increasing the size of the dots on the screen. Vertically, this is done by repeating the same display line. The number of repeat times is determined by the display zoom parameter. Horizontally, zoom is accomplished by extending each display word cycle and displaying fewer words per line, according to the zoom factor.

PRELIMINARY

82730
TEXT COPROCESSOR

- High Quality Display for Text and Graphics Applications
- Provides Proportional Spacing, Simultaneous Superscript/Subscript, Soft Font Support and Bit Map Graphics
- High Performance Manipulation of Text/Graphics Strings
- Programmable Bus Interface Handles 8 or 16 Bit Data and 16 or 32 Bit Addressing; iAPX 86/88/186/188 Compatible
- On-Chip Processing Unit Simplifies Software Design by Executing High Level Commands and Supporting Linked List Data Structures
- Extremely Flexible; Programmable Features Include Screen and Row Formats, Two Cursors, Character and Field Attributes and Smooth Scrolling
- Supports Multiple Windows
- High Resolution Display; Up to 200 Characters/Graphics Cells per Row and 2048 Scan Lines per Frame
- Separate Bus and Video Clocks Allow Optimization of Overall System Performance
- Provides a Complete LSI Solution for Display Control when Used in Conjunction with the 82731 Video Interface Controller

The 82730 Text Coprocessor is a high performance VLSI solution for raster scan text and graphics displays. The 82730 works as a coprocessor and has processing capabilities specifically tailored to execute data manipulation and display tasks. It provides the designer the ability to functionally partition his system thereby offloading the system CPU and achieving maximum performance through concurrent processing. The 82730 supports the generation of high quality text displays through features like proportional spacing, simultaneous superscript/subscript, dynamically reloadable fonts and user programmable field and character attributes. It supports high quality graphics with fast manipulation and display of bit map strings. An intelligent system interface and efficient software capabilities makes 82730 based systems easy to design.

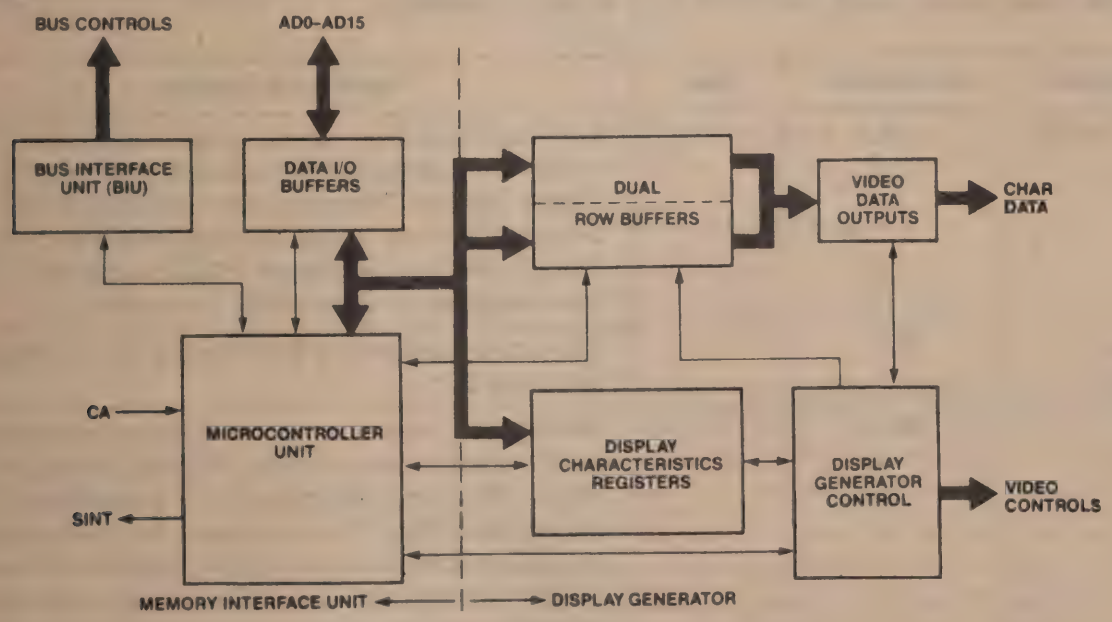


Figure 1. 82730 Block Diagram

Intel Corporation Assumes No Responsibility for the Use of Any Circuitry Other Than Circuitry Embodied in an Intel Product. No Other Circuit Patent Licenses are Implied.
©INTEL CORPORATION, 1984

AUGUST 1984
ORDER NUMBER: 210931-004

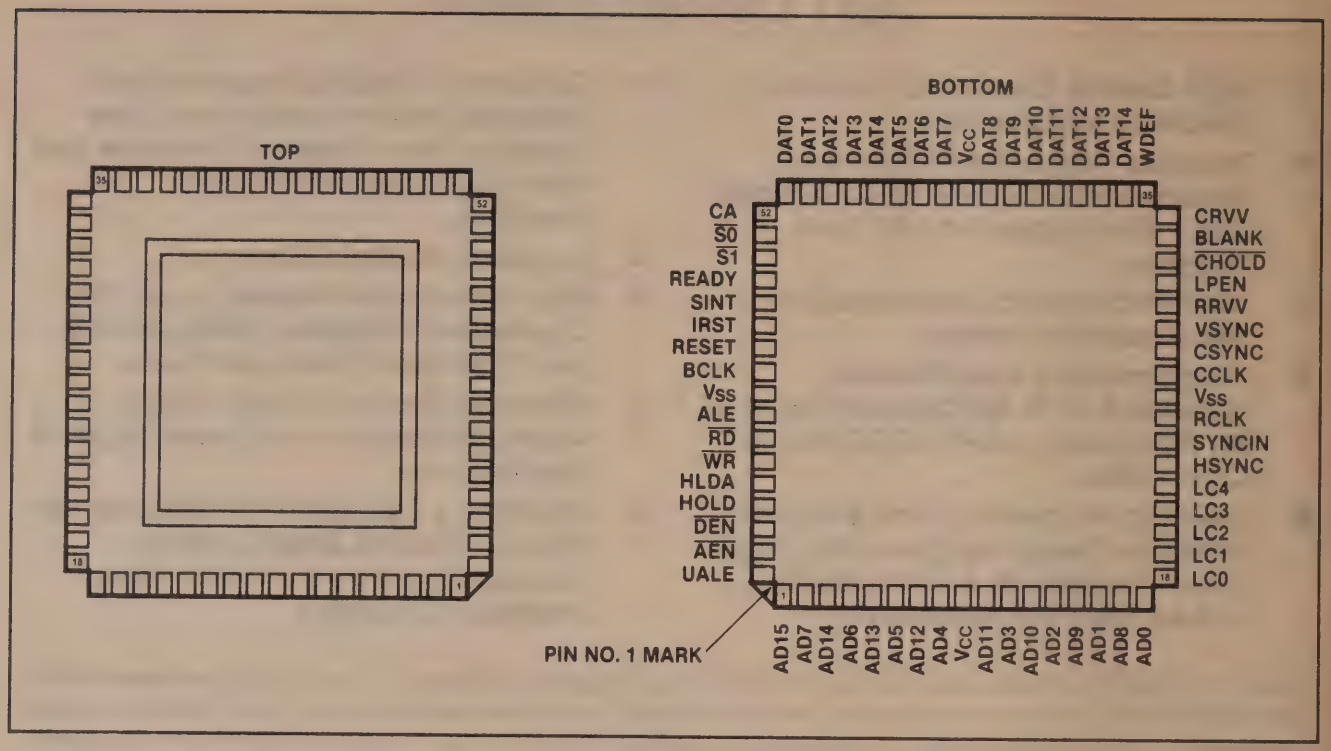


Figure 2. 82730 Pinout Diagram

Table 1. 82730 Pin Description

The 82730 is packaged in a 68 pin JEDEC Type A ceramic package.

Symbol	Pin Number	Type	Name and Function
AD15–AD0	1–8 10–17	I/O	Address Data Bus; these lines output the time multiplexed address (TU, T1 states) and data (T2, T3, T4 and TW) bus. The bus is active HIGH and floats to 3-state OFF when the 82730 is not driving the bus (i.e. HOLD is not active or when HOLD is active but not acknowledged, or when RESET is active).
BCLK	59	I	Bus clock; provides the basic timing for the Memory Interface Unit.
$\overline{RD}$	62	O	Read strobe; indicates that the 82730 is performing a memory read cycle on the bus. $\overline{RD}$ is active low for T2, T3 and TW of any read cycle and is guaranteed to remain high in T2 until the address is removed from the bus. $\overline{RD}$ is active low and floats to 3-state OFF when 82730 is not driving the bus. $\overline{RD}$ will return high before entering the float state and will not glitch low when entering or leaving float.

210931-004

Table 1. 82730 Pin Description (Continued)

Symbol	Pin Number	Type	Name and Function
$\overline{WR}$	63	O	Write strobe; indicates that the data on the bus is to be written in a memory device. $\overline{WR}$ is active for T2, T3 and TW of any write cycle. It is active LOW and floats when 82730 is not driving the bus. $\overline{WR}$ will return high before entering the float state and will not glitch low when entering or leaving float.
ALE	61	O	Lower Address Latch Enable; provided by the 82730 to latch the address into an external address latch such as 8282/8283 (active HIGH). Addresses are guaranteed to be valid on the trailing edge of ALE.
UALE	68	O	Upper Address Latch Enable; it is similar to ALE except that it occurs in upper address output cycle (TU).
$\overline{AEN}$	67	O	Address Enable; $\overline{AEN}$ is active LOW during the entire period when 82730 is driving the bus. It can be used to unfloat the outputs of the Upper and Lower Address latches.
$\overline{DEN}$	66	O	Data enable; provided as a data bus transceiver output enable for transceivers like the 8286/8287. $\overline{DEN}$ is active LOW during each bus cycle and floats when 82730 is not driving the bus. $\overline{DEN}$ will not glitch when entering or leaving the float state.
$\overline{S0}$, $\overline{S1}$	53, 54	O	Status pins; encoded to provide bus-transaction information: $\overline{S1}$ $\overline{S0}$ Bus Cycle Initiated 0 0 - - - (Reserved) 0 1 Memory Read 1 0 Memory Write 1 1 Passive (No bus cycle) These pins are directly compatible with iAPX 86, 186 status outputs $\overline{S1}$ and $\overline{S0}$. The status pins are floated when 82730 is not driving the bus. They will not glitch when entering or leaving the 3-state condition.
READY	55	I	READY; signal to inform the 82730 that the data transfer can be completed. Immediately after RESET, READY is asynchronous (internally synchronized) but can be programmed during initialization to bus synchronous.

Table 1. 82730 Pin Description (Continued)

Symbol	Pin Number	Type	Name and Function
HOLD	65	O	HOLD; indicates that the 82730 wants bus access. HOLD stays active HIGH during the entire period when 82730 is driving the bus.
HLDA	64	I	Hold Acknowledge; indicates to 82730 that it is granted the bus access as requested. HLDA may be asynchronous to 82730 clock. If HLDA goes inactive (LOW) in the middle of an 82730 bus cycle, the 82730 will complete the current bus cycle first, then it will drop HOLD and float address and bus control outputs.
CA	52	I	Channel Attention; used to notify 82730 that a command in the command block is waiting to be processed. CA is latched on its falling edge.
SINT	56	O	Status Interrupt; used to inform the processor that an unmasked interrupt has been generated in the 82730 status register.
IRST	57	I	Interrupt Reset; SINT is cleared by activating the IRST pin.
RESET	58	I	Reset; causes 82730 to immediately terminate its present activity and enter a dormant state. The signal must be active HIGH for at least 4 BCLK cycles and is internally synchronized to the bus clock.
CCLK	27	I	Character clock; input used to clock row buffer data, attribute, cursor and line count out of 82730. When more than one 82730 is connected in cluster mode, CCLK is used to synchronize output from both master and slave chips. A character data word will be output at every rising edge of CCLK.
RCLK	25	I	Reference clock; input used to generate timings for the screen layout and to define screen columns for data formatting. All raster output signals are specified relative to the rising edge of RCLK.
DAT0-DAT14	36-42 44-51	O	Video data bus output; the least significant 15 bits of the character data words are passed through the 82730 row buffer and made available on the pins DAT0-DAT14. The user has the flexibility to partition the data word into character and attribute bits per his requirements. The bits that are assigned for internally generated attributes may also be available at pin DAT0-DAT14. New character data will be shifted to these output pins at every rising edge of the CCLK. Together with LC0-LC4, they may be used to address the character generator or as attribute controls.

Table 1. 82730 Pin Description (Continued)

Symbol	Pin Number	Type	Name and Function
WDEF	35	O	Width Defeat; is used to indicate when the character is allowed to be a variable width or must be of fixed width. WDEF is LOW if the character being output is normal, but is HIGH if it is a superscript/subscript character or visible attribute (TAB or GPA). Optionally, WDEF can be held high by user command.
LC0-LC4	18-22	O	Line count outputs; used to address the character generator for the line positions in a row. The line number output is a function of the display mode and character attributes programmed by the user.
CSYNC	28	O	CCLK synchronization output; used to synchronize external character clock generator to reference clock timing. This output is active (high) outside the display field.
$\overline{\text{CHOLD}}$	32	O	CCLK Inhibit output; used by external logic to inhibit CCLK generation. This output is active (low) during the tab and end-of-row function.
SYNCIN	24	I	Synchronization input; used to synchronize the vertical timing counters to an externally generated VSYNC signal. Used by slave mode 82730 to synchronize to a master mode 82730 and by the master 82730 to lock the frame to an external source such as the power line frequency.
HSYNC	23	O (MASTER) I (SLAVE)	Horizontal Sync; in master mode, it is used to generate the CRT monitor's horizontal sync signal. It is active HIGH during the programmed horizontal sync interval. In interlace slave mode it is used in conjunction with SYNCIN to indicate the start of the even field for timing counter reset. At RESET, pin is set as an output in the LOW state.
VSYNC	29	O	Vertical Sync; active HIGH during the programmed vertical sync interval and used to generate the CRT monitor's vertical sync signal.
BLANK	33	O	Blanking output; used to suppress the video signal to the CRT. BLANK is clocked by CCLK.
CRVV	34	O	Character Reverse Video (CCLK output); used to externally invert video data output. CRVV is clocked by CCLK.
RRVV	30	O	Reference Reverse Video (RCLK output); to externally invert video in the field and border area if so programmed by user. It is LOW outside the border area, RRVV is clocked by RCLK.

Table 1. 82730 Pin Description (Continued)

Symbol	Pin Number	Type	Name and Function
LPEN	31	I	Light Pen Input; used to latch the position of a light pen. At the rising edge of this input, the column position and the row position of the 82730 will be loaded into the LPENROW and LPENCOL locations in the Command block.
V_{CC}	9, 43		Power; +5 volts nominal potential.
V_{SS}	26, 60		Power; ground potential.

FUNCTIONAL DESCRIPTION

Figure 1 shows a basic block diagram of the 82730 Text Coprocessor. The chip is divided into two main sections, the Memory Interface Unit and the Display Generator. The Memory Interface Unit controls fetching of the data and commands and handles interrupts and status. The Display Generator takes the data fetched by the Memory Interface Unit and presents it to the Video Interface logic which in turn drives the CRT monitor.

Memory Interface Unit

The Memory Interface Unit is divided into two sections: the Bus Interface Unit and the Microcontroller Unit. The Bus Interface Unit does the actual interfacing to the memory bus. It fetches or writes data under the control of the Microcontroller Unit. The Microcontroller Unit is a microprogrammed controller which is designed to efficiently fetch data from memory (up to 4 Mbytes/sec), and decode and execute various control and data handling commands. The Bus Interface Unit may be configured for 8 or 16 bit bus operation. With 8 bit bus selection, the user may specify either 8 or 16 bit character data. It also handles address manipulation automatically after being loaded from the Microcontroller Unit.

Display Generator

The Display Generator takes the data fetched from memory plus the modes programmed into it at initialization and produces all the video timing and the data transfers to support the CRT monitor at the character level. The 82730 works with an external character generator and the 82731 Video Interface Controller. The data is passed to the Display Generator from the Memory Interface Unit through the dual row buffers (similar in operation to the one in the 8275 CRT controller). The row buffers allow the user to use cheaper and slower main memory for display needs, provide on-chip attribute and display function generation, and avoid the conflict of access to the display memory (that would otherwise take place) by using an ordinary DMA access mechanism.

SYSTEM BUS INTERFACE

The Memory Interface Unit provides communication with system processor as well as memory interactions. Communication between the processor and the 82730 is performed via messages placed in communication blocks in shared memory. The processor can issue commands by preparing message blocks and directing the 82730's attention to them by asserting a hardware channel attention. The 82730 can cause interrupts on certain conditions, if enabled by the processor by activating its System Interrupt output, with status and error reporting taking place through the communication block in memory.

BUS INTERFACE UNIT:

The 82730 Bus Interface Unit provides an 8086 compatible bus interface which consists of:

- a 16/32 bit multiplexed Address/Data Bus: AD_0 - AD_{15}
- A complete set of local bus control signals compatible with 8086 min mode: $\overline{RD}$, $\overline{WR}$, ALE, $\overline{DEN}$ and READY
- Two status signals $\overline{S0}$ and $\overline{S1}$, compatible with 8086 max mode so that a bus controller (8288) can be shared for Multibus® access.
- Local bus arbitration through HOLD/ HLDA
- Two upper Address Latch controls: UALE and $\overline{AEN}$

The BUS INTERFACE UNIT (BIU) utilizes the same Bus structure as the 80186 or basically the same bus structure as the 8086 in both Min. and Max. mode, (with the exception of RQ/GT) and it performs a bus cycle only on demand (e.g., to fetch a command from the command block, or fetch a character from display data memory). The same set of T- states (T1, T2, T3, T4 and TW) of 8086 are used to handle the time multiplexed address/data bus. However, adaptations are made to handle 32 bit addresses as explained in the following sections where specific details of the BIU operation are described. Those details not mentioned can be assumed to be the same as those of the 80186.

ADDRESS BUS

The 82730 can be programmed during initialization to operate on either 16 bit or 32 bit (including any length between 17 and 32) physical addresses. Note that the 82730 does not use memory segmentation. The programmer must calculate physical addresses from segment and offset values to manipulate data structures.

To support 32 bit physical addresses with a 16 bit physical bus, multiplexing is again used. An upper address output cycle, TU, is inserted between T4 and T1 to output the upper 16 bits of address. The upper address latch enable, UALE, is used to latch the upper addresses during TU. Figure 3 shows the configuration of a 32 bit address bus.

TU occurs only when the 32 bit mode is specified and the upper address register of BIU is reloaded by MCU. This may result from:

i) Initialization

ii) Manipulation of display data or command pointers, for example, when a new string pointer is loaded during the execution of the END OF STRING command.

iii) DMA address incrementing across a 64K byte segment boundary.

iv) Regaining the bus after losing it to a higher priority master.

Timing of UALE is identical to that of ALE. $\overline{AEN}$ is equivalent to the active period of 82730 driving the bus.

If 16 bit address mode is programmed, TU will never occur in any bus cycle since the MIU treats all display pointers as 16 bit quantities and loading of internal upper address register is bypassed during address calculation. UALE always stays inactive, but $\overline{AEN}$ still goes active to indicate the 82730 has control of the bus.

DATA BUS

The 82730 is capable of operating on either an 8 bit or a 16 bit Data bus, as programmed during initialization on the SYSBUS byte.

When an 8 bit data bus is specified, the address present on AD15 to AD8 Address/Data lines is maintained for the complete bus cycle. Therefore, compatibility with 80188, 8088, 8089 and 8085 multiplexed address peripherals is maintained. Since the internal processing of the 82730 generally operates on 16 bit data quantities, two Bus fetch cycles are performed for each 16 bit data item. The first cycle fetches the low order byte, the second cycle the high order byte. These 2 fetch cycles are always executed back to back. If HLDA drops during the first cycle, the 82730 will not respond until the second cycle is completed. An 8 bit data mode can be selected in an 8 bit bus system that requires only 8 bit character data be fetched.

In 16 bit bus system, the 82730 requires all 16 bit quantities to start on even address boundary. Word transfer to or from odd boundary is not allowed since this type of transfer not only doubles the use of bus bandwidth but also can be easily avoided in application software. All that is required is to make sure all address pointers be an even number (A0=0).

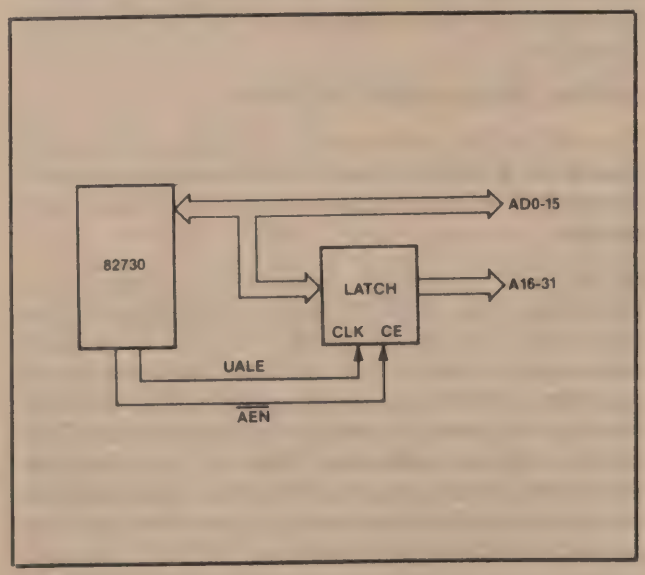


Figure 3. Address Extension up to 32 Bits

210931-004

BUS CONTROLS

The 82730 BIU provides both the 8086 MIN. Mode (Local Bus Control) and MAX. mode bus control signals simultaneously in any bus cycle. By providing a complete set of Local Bus control signals, the component count of the Local processing module is minimized.

Because only two types of Bus operations, Memory Read and Memory Write, are executed in the 82730 BIU, the 8086's $\overline{S2}$ status signal is omitted from the Max. mode controls. S2 could be set to "1" during any 82730 Bus cycle. $\overline{AEN}$ can be used to produce $\overline{S2}$ since it stays active whenever 82730 is driving the bus. The status signals become valid at the middle of the cycle before T1 which could be either T4 or TU.

$\overline{BHE}$ is not provided on the 82730 because, the 82730 only writes words to even address boundaries and bytes to the upper byte position. For these writes $\overline{BHE}$ is always high. A pullup resistor or a three-state buffer controlled by $\overline{AEN}$ can provide this signal.

DT/R is also not provided on the 82730 because its function can be replaced with $\overline{S1}$, latched by ALE.

After RESET is applied, READY is set to be an asynchronous input. An on-board synchronization circuit provides reliable operation for any type of system. During initialization, READY may be programmed to be bus synchronous. For those systems that can meet the set-up time specifications, this mode provides more efficient bus utilization.

LOCAL BUS ARBITRATION

The 82730 BIU is designed to function as a bus master in a multimaster Local bus environment using the HOLD/HLDA protocol for Bus arbitration.

In the Self Contained Arbitration scheme, one processor and one 82730 share access to the local bus. The 82730 raises its HOLD request whenever it needs bus access. After HLDA is granted from the processor, the 82730 will not start driving the bus until 2 clock cycles later. This latency allows sufficient time for the 8086 or 80186 processor to get off the bus. When 82730 completes its bus accesses, it will first float its output drivers before dropping the hold request.

In a Local bus configuration with three or more bus masters, a higher priority DMA Peripheral device can preempt the HLDA from a 82730 which is the current bus master. The 82730 will complete its current bus cycle, then float its output drivers and drop the HOLD request. However, the 82730 may raise the HOLD request again 2 clock cycles later if it still needs the bus to complete the interrupted burst DMA activities.

DMA BURST AND SPACE

Some system configurations using the 82730 would be adversely affected by the long burst data transfers which the Memory Interface Unit (MIU) may occasionally desire. Since the 82730 will normally be configured as one of the higher priority bus masters, burst lengths must be limited for these systems. For this reason, the length of a burst transfer and the number of memory cycles between burst transfers are both programmable via the mode registers:

	15	14	8	7	6	0
MPTR	—	BRSTLEN		—	BRSTSPAC	

BRSTLEN - Burst Length. Determines the number of contiguous word-fetch cycles which may be requested. Programmable from 1 to 127. Note that in an 8 bit bus, 16 bit data system, the burst counter only increments once for the 2 bus cycles required to complete a word fetch. (Note: burst length = 0 is not defined and should not be programmed with a non-zero burst space)

BRSTSPAC - Burst Space. Determines a minimum number of bus clocks to occur between burst accesses. Programmable from 0-511 in increments of four. Zero space selects an infinite burst length.

A DMA burst could be terminated before the programmed burst length is reached in the following circumstances:

i) The MIU does not need any more bus accesses, for example, when the row buffer is filled.

ii) A datastream command is encountered and the MIU must execute the command first before it resumes data accessing.

iii) The bus is taken away by a higher priority device in multi-master bus configuration.

In these cases, the burst counter is cleared. The BIU must complete a full burst before it waits through the SPACE cycles. DMA Burst/Space will be set to zero space until the completion of the first MODESET command.

210931-004

INITIALIZATION OF BIU

Upon activation of the RESET input, the 82730 BIU will stop all operations in progress and deactivate all outputs. It will stay in this quiescent state until memory access is requested by the MCU after MCU receives its first channel attention after RESET. The following table shows the state of all MIU outputs during and after reset.

Table 2. 82730 Bus During and After Reset

Signals	Condition
AD15-0	Three-state
$\overline{RD}$, $\overline{WR}$, $\overline{DEN}$	Driven to '1' then three-state
$\overline{S0}$, $\overline{S1}$	Driven to '1' then three-state
ALE, UALE	Low
$\overline{AEN}$	High
HOLD	Low
SINT	Low

82730 COMPATIBILITY ISSUES

82730 Bus Clock Compatibility

The 82730 uses the 50% duty cycle output of the iAPX-186 at 8 MHz or that generated by a clock generator such as the 82285. A different duty cycle clock may be used at lower frequencies, so the 82730 is also useable with the iAPX-86, 88 family.

82730 Bus Interface Compatibility

The bus interface compatibility between the 82730 and another bus master has four main issues: data bus width, addressability, control bus structure and local bus mastership arbitration.

Data Bus

Data Bus width compatibility with all 85/86 family processors (8085, 8086, 8088, 80188, 80186, and 80286) is being supported by the 8/16 data bit programmability already discussed. This allows interfacing to the above processors either directly or through a Multibus-like interface.

Address Bus

The 82730 uses real 32-bit addresses. The user's software must calculate real addresses; this general addressing scheme allows the 82730 to be used with any microprocessor.

Control Bus

The 82730 implements both 8086 minimum and maximum mode bus control structures. This was done to maximize compatibility with the 80186 which has the same structure. This allows the 82730 to be run locally (minimum mode) with a 8085, 8086, 8088, 80188, or 80186. The 80186/188 and 82730 can run together at 8MHz because of clock duty cycle considerations. The 82730 can only communicate to an 80286 via a system bus (such as MULTIBUS), bus interface, or dual-port RAM.

INITIALIZATION SEQUENCE

The first CA (Channel Attention) after Reset causes an Initialization Sequence to be executed. The system processor must set up the appropriate initialization information in memory and set the BUSY flag in the Intermediate Block to a non-zero value prior to issuing this CA.

Initially, 32-bit addressing and 8-bit data bus width are assumed until the corresponding information is fetched during the initialization. First the SYSBUS byte is fetched from memory location FFFF FFF6. (When the address bus is less than 32 bits wide, the higher order bits are unused.) The format for SYSBUS byte is shown in Figure 4 and is the same as that used for 8089. The data bus width is specified by the least significant bit w, with w=0 indicating an 8-bit bus and w=1 signifying a 16-bit bus.

A 32-bit real address pointer is then fetched from memory locations FFFF FFFC through FFFF FFFF, with lower bytes of the pointer residing in lower addresses. This pointer is used as an Intermediate Block Pointer (IBP).

The Intermediate Block Pointer (IBP) is incremented by two and is used to locate the Command Block Pointer (CBP). Four bytes are fetched irrespective of whether a 16-bit or 32-bit addressing option is used. The System Configuration byte (SCB) is then fetched from location (IBP + 6).

The least significant bit, (U of the SCB) specifies 16 or 32-bit addressing option, with U=0 indicating 16 bit addressing and U=1 specifying 32-bit addressing. The SCB also contains information about cluster operation. Since up to four 82730's can be connected in a cluster with their respective data interleaved in memory, cluster information is needed for the data access task. The SCB specifies Cluster Number (CL NO), which is the number of 82730's connected in a cluster and Cluster Position (CL POS) which is the position

210931-004

of this particular 82730 within the cluster. CL NO = 0, 1, 2 or 3 indicates a cluster containing 1, 2, 3 or 4 82730's respectively. Similarly, CL POS = 0, 1, 2 or 3 indicates 1st, 2nd, 3rd or 4th position respectively. Each 82730 adds an offset equal to 2 * CLPOS to the SPTR fetched from memory and increments the pointer by 2 * (CL NO + 1). The programming of CL NO and CL POS is independent. No checking is done for CL POS greater than CL NO on the 82730. Note that at least one 82730, in a cluster (even if it is a cluster of one), must be assigned as cluster position zero (CL POS = 0) for Virtual Display mode to work properly.

7							0	
0	0	0	0	0	0	0	W	SYSBUS Byte

W	Data Bus Width
0	8-Bit
1	16-Bit

7	6	5	4	3	2	1	0	
SRDY	DTW16	M/S	CL	POS	CL	NO	U	SCB Byte

SRDY	READY MODE
0	Asynchronous
1	Synchronous

DTW16	Display Data Mode
0	8-bit data
1	16-bit data

M/S	Mode
0	Slave
1	Master

CL POS	Position in Cluster
00	1st
01	2nd
10	3rd
11	4th

CL NO.	No. of 82730's In Cluster
00	1
01	2
10	3
11	4

U	ADDR BUS WIDTH
0	16-bit
1	32-bit

Figure 4. SYSBUS and SCB Encoding

The SCB also contains an M/$\overline{S}$ bit which specifies a master or slave mode. The M/$\overline{S}$ bit is stored internally for use by the Display Generator (DG). M/$\overline{S}$ = 1 indicates a master mode and M/$\overline{S}$ = 0 specifies a slave mode. The format for the System Configuration Byte (SCB) is shown in Figure 4. Following these actions, the BUSY flag in the Intermediate Block at address IBP is cleared and a normal Channel Attention sequence is then executed.

The last two bits in the SCB are DTW16 and SRDY. DTW16 specifies whether the display data in 8 bit bus mode (W=0) is 8 or 16 bit. If a 16 bit system is specified (W=1) then DTW16 is ignored and forced internally to a "one". SRDY specifies whether the clock synchronization circuit for the READY pin is internal (SRDY=0) or external (SRDY=1).

The Initialization Control Blocks in memory are illustrated in Fig. 5a. How these fit into the control structure of the 82730 is shown in Figure 5b.

Channel Attention Sequence

When the processor activates CA, an internal latch in 82730 is set on the falling edge of CA input and this latch is sampled by the MCU. The first CA activation after reset causes the 82730 to execute an initialization sequence. Any subsequent activation will cause the MCU to start processing the command block by fetching a channel command.

If a display is in progress, the MCU will sample CA at each end of frame, otherwise it will sample CA every cycle until it is found active. When CA is found active, the MCU will fetch the command byte from "COMMAND" location in the command block, execute the command and clear the BUSY flag upon completion. The internal CA latch is also cleared by the MCU. An invalid command code has the effect of NOP and the BUSY flag is cleared. It will also cause the Reserved Channel Command (RCC) status bit to be set.

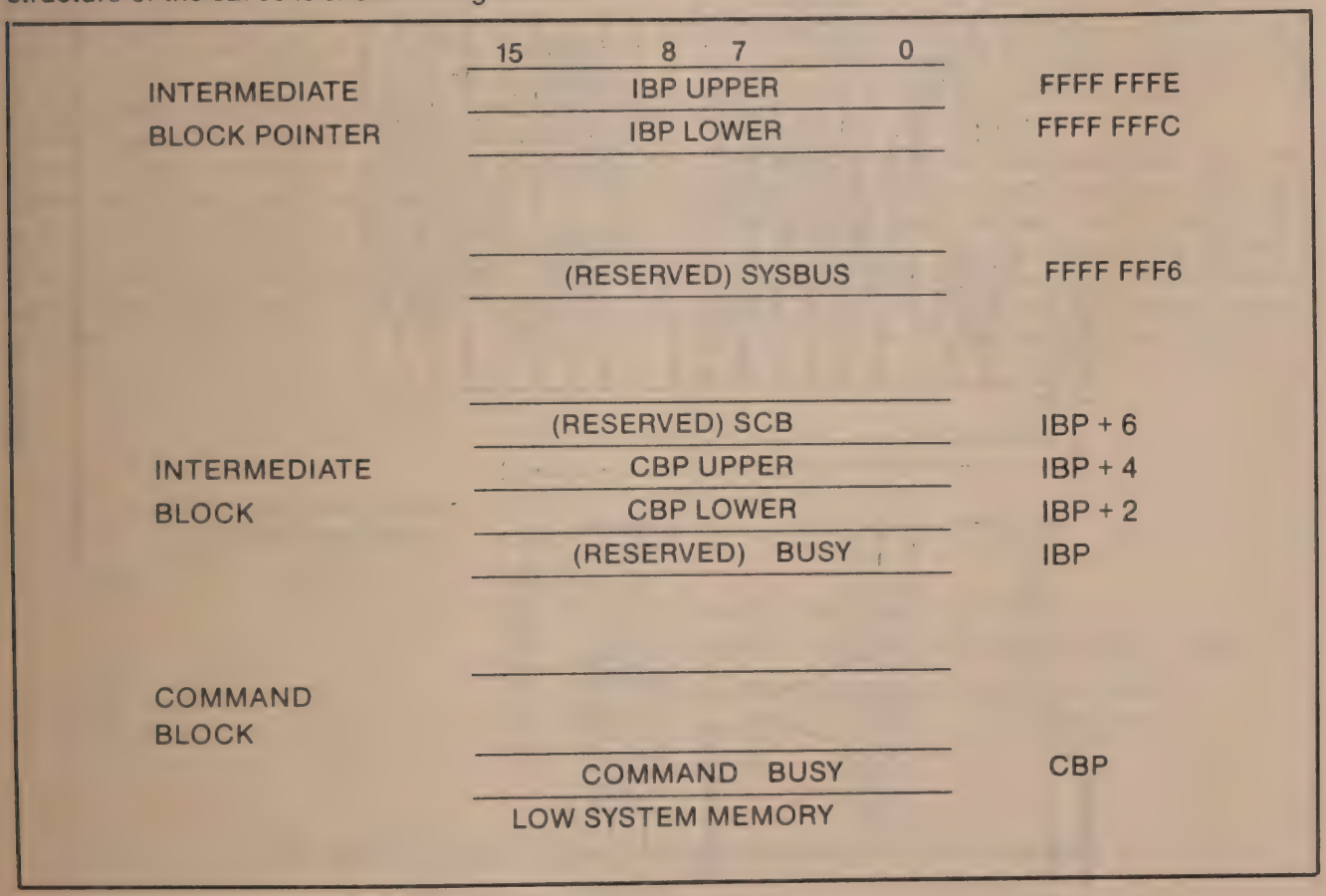


Figure 5a. Initialization Control Blocks

210931-004

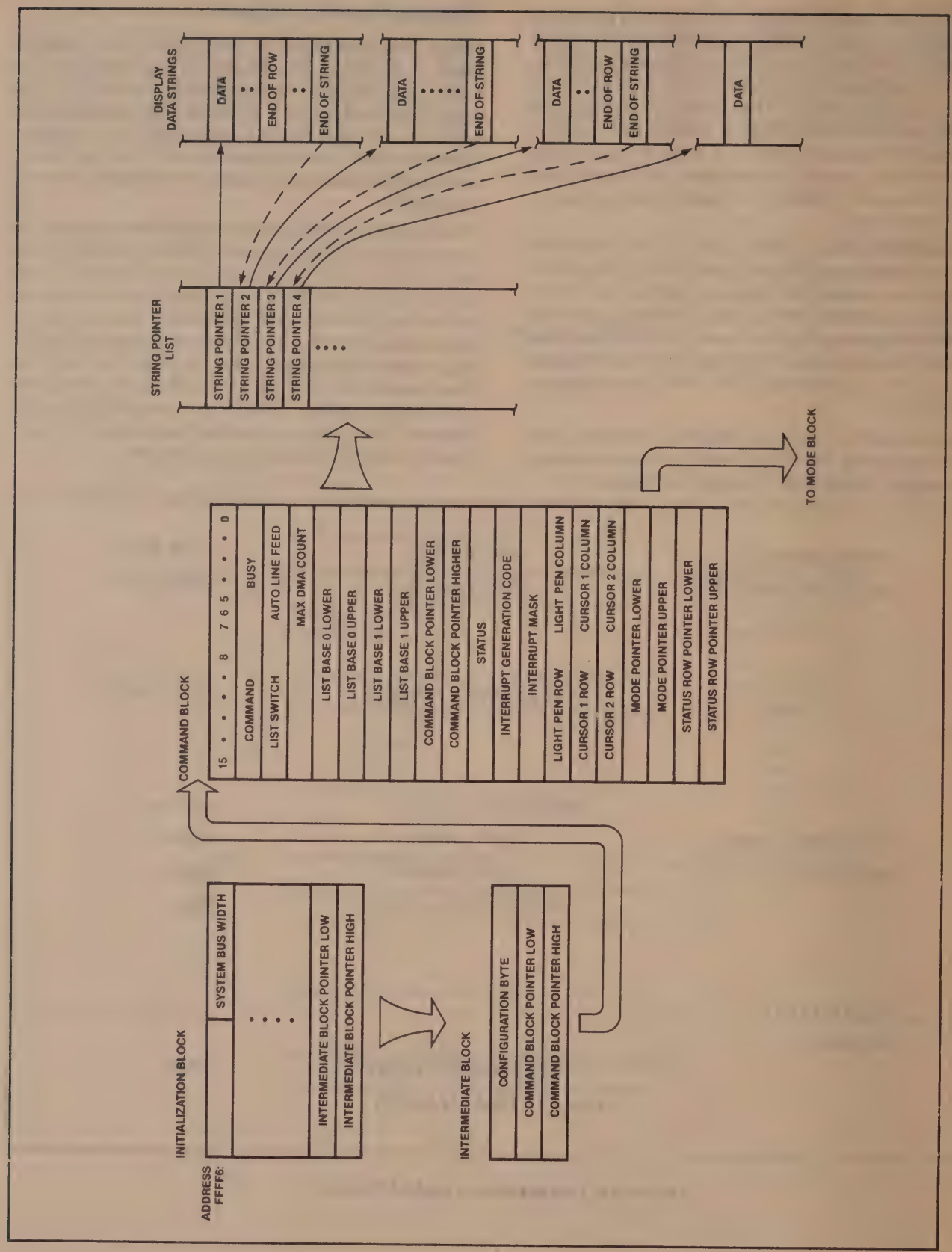


Figure 5b. Control Structure of the 82730

82730 TEST FEATURES

The 82730 has built in Self-Test features that provide testability at the component or at the board level. These features include the test commands and the output pin force capability and are described below.

Output Pin Force Capability

A capability to force logic states (high, low, high impedance) on all output pins is provided in the 82730 Text Co-Processor. This is accomplished by providing a stimulus on pins LC0-LC2 during chip reset. This feature is used for dc parametric tests on the output pins.

The state of pins LC0-LC2 is monitored during chip reset. The state of these pins is latched internally on the falling edge of chip reset. If no external inputs are applied during reset, the state observed will be all 1's and no action will be taken by the 82730. If any external inputs are applied to pins LC0-2 during reset, the resulting action will depend upon the state latched on the falling edge of reset. The 82730 maintains pins LC0-LC2 in high impedance state for the duration of chip reset to avoid contention with external inputs. Also internal pull-ups ensure that a state of all 1's will be detected if no external inputs are applied.

The actions corresponding to each of the observed states of pins LC0-LC2 are summarized in Fig. 6a.

Stand-Alone Self Test

The built-in Self Test capability of the 82730 can be invoked in a stand-alone mode by applying an external stimulus through pins LC0-LC2 during chip reset. This is the same mechanism as the one used for forcing logic states on output pins. Fig. 6a.

If pin LC2 is pulled low during chip reset, the 82730 executes a built-in self test. Upon completion of the self-test, a 16-bit signature, generated internally as the test result, is output via pins WDEF, DAT14-DATO. The completion is signalled by providing a logic "O" output on pin LC3 as a completion flag. The signature will remain on the output pins until the next chip reset. The 82730 will enter an idle state awaiting chip reset and will not respond to any external inputs until a reset signal is applied. During the process of presenting the signature onto WDEF, DAT14-DATO, the signature will also appear briefly on the AD bus in the form of a bus cycle with two 8-bit accesses to addresses, AAAAH, AAABH. However, this phenomenon is only incidental. Pins WDEF, DAT14-DATO should be used for observing the signature.

The stand-alone self test includes the testing of internal address pointer registers. These registers are not tested when the self test is invoked by issuing a "Self Test" command. (See under Channel Commands below). Therefore, the signature generated during stand-alone self test will be different from that generated by the "Self Test" command.

State of Pins LC0-LC2 During Chip Reset			Action
LC2	LC1	LC0	
0	X	X	Invoke Stand-Alone Self Test
1	0	0	Force all Outputs to High Impedence State
1	0	1	Force all Outputs to Logic High State
1	1	0	Force all Outputs to Logic Low State
1	1	1	NOP

Figure 6a. Output Pin Forcing and Stand-Alone Self Test Invocation

82730 CHANNEL COMMANDS

Table 3. Channel Commands

	COMMAND	OP CODE		
1	START DISPLAY	0000	0001	01 H
2	START VIRTUAL DISPLAY	0000	0010	02 H
3	STOP DISPLAY	0000	0011	03 H
4	MODE SET	0000	0100	04 H
5	LOAD CBP	0000	0101	05 H
6	LOAD INTMASK	0000	0110	06 H
7	LPEN ENABLE	0000	0111	07 H
8	READ STATUS	0000	1000	08 H
9	LD CUR POS	0000	1001	09 H
10	SELF TEST	0000	1010	0A H
11	TEST ROW BUFFER	0000	1011	0B H
12	NOP	0000	0000	00 H
13	(RESERVED)	From: 0000 To: 1111	1100 1111	0C H FF H

The system processor issues channel commands to 82730 via the Command Block. The processor first checks if the BUSY flag in the command block has been cleared. It should wait for the BUSY flag to be cleared before proceeding with the issuing of a command. When the BUSY flag is cleared, the processor places a command byte in the "COMMAND" location in command block, sets the BUSY flag to a non-zero value and asserts Channel Attention (CA), by activating the CA input to 82730. A Channel Attention should not be issued, if the BUSY flag has not been cleared.

START DISPLAY

0000	0001	CMD Byte

LISTSWITCH, Auto Linefeed, Max DMA Count and Cursor Position values are fetched from the Command Block and stored internally after this command is received. The BUSY flag is cleared and the normal display process is activated.

The MCU fetches strings of data from the memory, using the parameters LISTSWITCH, LBASE0 and LBASE1. The data fetched is interpreted as data-stream commands or character data to be displayed by the Display Generator. The MCU loads the data into one of the two Row Buffers in the CRT controller, while the Display Generator displays the data from the other buffer, the buffers being swapped at the end of the row. Any data-stream commands encountered during data fetch are immediately executed.

The display process is continued until it is deactivated by a STOP DISPLAY command or a Reset. Other channel commands can be issued while a display is in progress and they will be executed when CA is found active at one of the periodic samplings at each end of frame.

The DIP (Display in Progress) status bit is set and the VDIP (Virtual Display in Progress) is cleared upon receiving a START DISPLAY command. Both bits are reset upon receiving a STOP DISPLAY command or a Reset.

It is necessary to load in proper mode information through a MODESET command before activating the display. Following Reset, START DISPLAY command will not be executed, i.e., will result in a NOP until a MODESET command has been issued.

START VIRTUAL DISPLAY

0000	0010	CMD Byte

LISTSWITCH, Auto Linefeed, Max DMA Count and Cursor Positions are fetched from the Command Block and stored internally upon receiving this command. The BUSY flag is cleared and the Virtual Screen display process is activated.

The operation of Virtual Screen display process is similar to that of a regular display process, except for following a different data access mechanism. The parameters LISTSWITCH, LBASE0 and LBASE1 in the command block represent ACCESS SWITCH, ACCESS BASE0 and ACCESS BASE1 respectively, in virtual screen display.

The VDIP (Virtual Display in Progress) status bit is set and the DIP status bit is cleared upon receiving a START VIRTUAL DISP command: Both DIP and VDIP are reset upon receiving a STOP DISPLAY command or a Reset.

START VIRTUAL DISPLAY command will not activate a display and results in a NOP until a MODESET command is issued after a Reset.

STOP DISPLAY

0000	0011	CMD Byte

The display process is deactivated upon receiving this command. The DIP and VDIP status bit are reset and the BUSY flag is cleared.

This command blanks the display. HSYNC and VSYNC are **not** affected.

MODESET

0000	0100	CMD Byte

The Mode Pointer contained in command block location (CBP + 30) is used to access the Mode Block and the modes are fetched sequentially and loaded into the corresponding internal registers in 82730. LISTSWITCH, Auto Linefeed, Max DMA Count and Cursor Positions are fetched from the Command Block and stored internally upon completion and the BUSY flag is cleared.

The organization of mode words in the mode block and the parameters supplied by them are shown below (See Figure 10). Some of these parameters which are critical to the operation of a text coprocessor are required to remain unchanged over most of normal operation. No provision is made to prevent MODESET from changing these parameters and it is left to the designer to insure that they are not changed.

The modes provide horizontal and vertical mode display parameters, interlace information, DMA burst and spacing specifications, cursor characteristics as well as attribute enables and bit-selects. Typically, this would be the first command issued after initialization. The Mode Block provides all the parameters needed for a complete initialization of the 82730 for display. Thus a single Modeset command can fully initialize the chip. Note that until the first Modeset command is sent, certain functions such as VSYNC and HSYNC are not enabled.

It is necessary to set up proper mode information, before activating a display. Therefore, a display activating commands should not be issued unless proper mode information has been loaded through a MODESET command. START DISPLAY and START VIRTUAL DISPLAY commands will result in a NOP if a MODESET command has not been issued since the most recent Reset.

LOAD CBP

0000	0101	CMD Byte

The address pointer "NEW CBP" contained in the command block is fetched and stored in the CBP register in the text coprocessor, replacing the old CBP. This effectively moves the command block in the memory. The Command byte from the new Command Block is fetched and the specified channel command is executed. The BUSY flag in the new Command Block is cleared upon completion.

LOAD INTMASK

0000	0110	CMD Byte

The interrupt mask contained in location "INT MASK" in the command block is fetched and stored internally in the CRT controller. When a particular mask bit is set, the interrupt is disabled for a status bit in the corresponding bit position. An interrupt is generated by the text coprocessor by activating the SINT pin, if a status bit is 1 and the corresponding bit in the interrupt mask is 0. The BUSY flag is cleared upon completion.

Interrupts can be enabled for the following status bits.

7	6	5	4	3	2	1	0	BIT
—	RDC	RCC	FDE	EOF	DBOR	LPU	DUR	STATUS WORD

RDC: Reserved Datastream Command Encountered
RCC: Reserved Channel Command Executed
FDE: Frame Data Error (Fetching characters past physical End of Frame)
EOF: End of "n" frames (Logical end of nth frame)
DBOR: Data Buffer Overrun (Row Buffer filled completely without encountering END OF ROW command)
LPU: Light Pen Update
DUR: Data Underrun (Buffer swap initiated before finishing Row Buf loading)

READ STATUS

0000	1000	CMD Byte

The internal status register is written to "STATUS" location in the command block. The status register is then cleared, however DIP and VDIP status bits are not cleared. LISTSWITCH, Auto Linefeed, Max DMA Count and Cursor Positions are fetched from the Command Block and stored internally. The BUSY flag is then cleared.

LD CUR POS

0000	1001	CMD Byte

The display row and column positions of cursors 1 & 2 as set in locations "CUR1 ROW," CUR1 COL," "CUR2 ROW" and "CUR2 COL" in the command block are loaded into internal registers in the CRT controller. Also LISTSWITCH Auto Linefeed and Max DMA Count are loaded from the Command Block and the BUSY flag is cleared. This command is used to change the cursors only. Note that the cursor positions are also updated with the execution of other channel commands.

The cursor characteristics for display are defined by the mode. During the display process, a cursor will be displayed accordingly at the position specified above.

STATUS WORD

15-9	8	7	6	5	4	3	2	1	0
—	VDIP	DIP	RDC	RCC	FDE	EOF	DBOR	LPU	DUR

LPEN ENABLE

0000	0111	CMD Byte

The Light Pen detection process is enabled to search for a rising edge on the LPEN pin. The BUSY flag is then cleared.

If the display process is active and a rising edge is detected on the LPEN input, the corresponding row and column position on the screen is stored internally. At the next end of frame, the LPEN position is written to locations "LPENROW" and "LPENCOL" in the command block and the LPU (Light Pen Update) status bit is set.

If the display process is not active, this command has no immediate effect. However, the LPEN detection process remains enabled and will take effect if a display is activated subsequently.

TEST COMMANDS

The test commands for the 82730 are issued in the same manner as the normal channel commands. However, the parameters used by test commands are different from those used by the channel commands in normal operation. Therefore, a Test Block which is similar in format to the Command Block is defined. Switching between Command Block and Test Block is accomplished using the "Load CBP" command. The Test Block differs only in the parameter locations associated

with the command. The locations for New CPB, command byte and busy flag are the same for both Command Block and Test Block. The "Test Result' location in Test Block corresponds to the "Status" location in Command Block.

The test commands can be executed, following chip reset, only until the first Modeset command is issued. Once a Modeset command has been executed following chip reset, any subsequent test commands will not be executed and will result in a NOP.

"Self Test" Command

0000	0010	CMD Byte

A built-in Self test is performed using an internal test pattern. The signature generated during the test is written to the Test Result location (TBP+18) in the Test Block. The Busy Flag in the Test Block is then cleared. The Self Test command must be immediately preceded by a chip reset in order to ensure a consistent signature.

The Test Block format for issuing the Self Test command is shown in Figure 6b.

"Row Buffer Test" Command

0000	1011	CMD Byte

The Load Pointer in Test Block is fetched. It points to the system memory area storing the test pattern to be used for testing the on-chip RAM (i.e. - the Row Buffers). The Store Pointer, which points to memory area where the data read back from the RAM will be written, is also fetched from Test Block.

Successive words are fetched from memory and written to the Row Buffer, until it is completely filled. Note that three extra words beyond the maximum Row Buffer capacity will be fetched. If N = Max Row Buffer capacity, (N+3) words will be fetched from memory. The extra words fetched will be ignored. The Row Buffer contents are then read back and are written to successive locations in memory area pointed to by the Store Pointer. The test is then repeated on the second Row Buffer. Note that the (N+4)th word in the pattern stored in memory constitutes the first word written to the second Row Buffer. The data storage for the Row Buffer test patterns is illustrated in Figure 6c.

	15 — 8	7 — BIT 0	TEST BLOCK POINTER (TBP)
	COMMAND	BUSY	
PARAMETER			TBP+2
LOCATIONS			TBP+4
NOT USED			TBP+6
FOR SELF TEST			TBP+8
COMMAND			TBP+10
			TBP+12
	NEW CBP LOWER		TBP+14
	NEW CBP UPPER		TBP+16
	TEST RESULT		TBP+18

Figure 6b. Test Block Format for "Self Test" Command
(For both 16-bit and 32-bit addressing modes)

Internally, the Row Buffers are 17-bits wide, while the data path is 16-bits wide. During the writing of data to Row Buffers, a complement of bit 15 is written to bit 16 of the Row Buffer in order to test all 17 bits. During the read back, two data words are stored in system memory for each location in the Row Buffer. The first word will consist of bits 0-15 read from the Row Buffer, while the second word will consist of bits 0-14 and bit 16 from the Row Buffer. Thus a total of 4*N words will be stored back in system memory as a result of the Row Buffer Test (2*N for each Row Buffer).

A signature is generated during the test and is written to Test Result location in Test Block upon completion. The BUSY flag in the Test Block is then cleared.

The Test Block format for issuing the Row Buffer Test command is illustrated in Figures 6d.1 and 6d.2. Note that the locations for Load Pointer and Store Pointer parameters are different for 16-bit and 32-bit addressing modes.

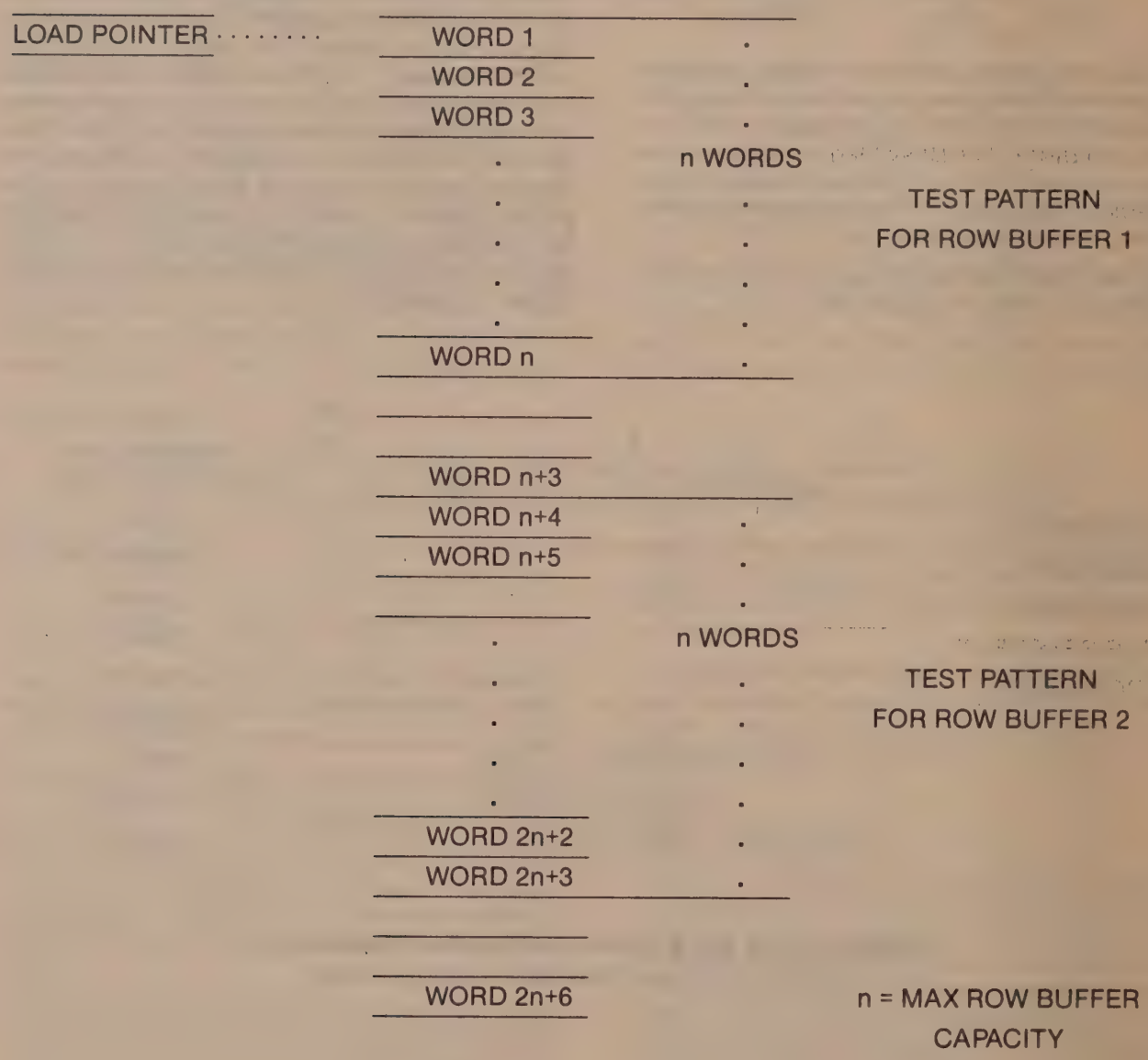


Figure 6c. Data Storage for Row Buffer Test Command

15 — 8	7 — BIT 0	TEST BLOCK POINTER (TBP)
COMMAND	BUSY	
(RESERVED)		TBP+2
LOAD POINTER LOWER		TBP+4
LOAD POINTER UPPER		TBP+6
STORE POINTER LOWER		TBP+8
STORE POINTER UPPER		TBP+10
(RESERVED)		TBP+12
NEW CBP LOWER		TBP+14
NEW CBP UPPER		TBP+16
TEST RESULT		TBP+18

Figure 6d.1 Test Block Format for "Row Buffer Test" Command
(32-bit addressing mode)

15 — 8	7 — BIT 0	TEST BLOCK POINTER (TBP)
COMMAND	BUSY	
(RESERVED)		TBP+2
(RESERVED)		TBP+4
LOAD POINTER		TBP+6
(RESERVED)		TBP+8
STORE POINTER		TBP+10
(RESERVED)		TBP+12
NEW CBP LOWER		TBP+14
NEW CBP UPPER		TBP+16
TEST RESULT		TBP+18

Figure 6d.2 Test Block Format for "Row Buffer Test" Command
(16-bit addressing mode)

NOP

0000	0000	CMD Byte

LISTSWITCH, Auto Linefeed, Max DMA Count, and Cursor Positions are fetched from the command block and stored internally as in all other channel commands. The Busy flag is then cleared.

82730 DATASTREAM COMMANDS

Datastream Commands

Datastream Commands are commands embedded in the data fetched from memory by the data access task. These commands are differentiated from character data by the command bit. The most significant bit (MSB) of each data word is designated as the command bit. If the command bit is "1", the lower 15 bits of the data word are interpreted as a datastream command, while if the command bit is "0" the lower 15 bits (or 7 bits if DTW16=0) are interpreted as character data.

Datastream Command Operation

During the data access task, the Micro Controller Unit (MCU) examines the command bit of each data word fetched. If the command bit is 1, it executes the datastream command specified in the data word. Otherwise, it stores the lower 15 bits of the data word in the Row Buffer as character data. This process is repeated for each data word fetched.

Datastream commands can be used for changing Row Characteristics on a row by row basis, for carrying out editing functions and for formatting data into rows and frames. These commands are executed by the MCU immediately after they are encountered. As a convenience for the user, the set of all possible command codes starting with "11" in the two most significant bits has been designated as NOP commands. The user can use these command codes for any desired purpose. All other command codes which are not presently defined, are reserved for future expansion and should not be used by the user. The currently undefined codes cause the RDC (Reserved Datastream Command) status bit to be set and also generate an interrupt, if enabled. Reserved command codes should not be used.

Datastream Command List

Table 4. 82730 Datastream Commands

COMMAND		COMMAND CODE		OP CODE
		OP CODE	PARAMETERS	
1	ENDROW	1000 0000	XXXX XXXX	80
2	EOF	1000 0001	XXXX XXXX	81
3	END OF STRING & END OF ROW	1000 0010	XXXX XXXX	82
4	FULROWDESCRPT	1000 0011	"n"	83
5	SL SCROLL STRT	1000 0100	XXX SCR LINE	84
6	SL SCROLL END	1000 0101	XXX END LINE	85
7	TAB TO n	1000 0110	"n"	86
8	LD MAX DMA COUNT	1000 0111	COUNT	87
9	ENDSTRG	1000 1000	XXXX XXXX	88
10	SKIP n	1000 1001	"n"	89
11	REPEAT n	1000 1010	"n"	8A
12	SUB SUP n	1000 1011	"n"	8B
13	RPT SUB SUP n	1000 1100	"n"	8C
14	SET GEN PUR ATTRIB	1000 1101	GPA OP	8D
15	SET FIELD ATTRIB	1000 1110	XXXX XXXX	8E
16	INIT NEXT PROCESS (Command process command)	1000 1111	XXXX XXXX	8F
17	(RESERVED)	10XX XXXX	XXXX XXXX	90-BF
18	NOP	11XX XXXX	XXXX XXXX	C0-FF

The preceding commands are recognized as valid datastream commands. The corresponding command codes are also indicated. It should be noted that the most significant bit of the command bit is always 1, in order for the word to be interpreted as command.

The "Init Next Process" command can be issued only through a command process in Virtual Screen Display. It is included in this list because its operation is analogous to a datastream command in a virtual screen access environment. Also, in virtual screen display certain datastream commands are interpreted differently, depending upon whether they are encountered in a process datastream or as command process commands. When a command is ignored (becomes a NO-OP) in a virtual display, any parameters that are associated with it are also ignored. The command process command operation is discussed separately. The operation of all other datastream commands is described below.

ENDROW

15	14	8	7	0
1	000	0000	XXXX	XXXX

This command signifies that no more characters will be loaded in the Row Buffer for this row and an End of Row indicator is stored accordingly. When the row currently being loaded is displayed, the Display Generator (DG) will blank the screen from the end of row character position until the physical end of row.

The Micro Controller Unit (MCU) stops fetching data and waits for DG to swap the Row Buffers. The data access task is resumed following the buffer swap. If a physical end of frame is reached while the MCU is waiting for a buffer swap the MCU ceases to wait and executes an EOF (End of Frame) command.

In virtual display, this command is interpreted as a VEOR (Virtual End of Row) if encountered in a virtual process datastream.

VEOR

ENDROW command in a virtual process datastream is interpreted as VEOR (Virtual End of Row) and it terminates a virtual row. The current LPTR is stored in the process header addressed by the "Process Addr" register. The Max Count register is also stored in the Max DMA Count location in the process header. Similarly, the Field Attribute Mask is also stored in the header. In addition, in auto linefeed mode (ALF = 1) other parameters characterizing the process state are also saved in the header. The "Process Addr" register is loaded with the address of the header of the next process fetched from the Access table. The "Access Tab Addr" register is post-incremented by two if a 16-bit addressing option is used and by four if 32-bit addressing is used. The data access task is then resumed for the next process.

EOF

15	14	8	7	0
1	000	0001	XXXX	XXXX

This command (End of Frame) signifies that no more characters will be loaded in the Row Buffers for this frame. The Micro Controller Unit (MCU) stops fetching data words and waits for the physical end of frame. If a virtual display is in progress, this command is interpreted as VEOS (Virtual End of Frame), if encountered in a virtual process datastream.

The Display Generator (DG) swaps the row buffers at the end of the current display row and starts displaying the row containing the EOF command. When the character preceding the EOF command is displayed, the DG blanks the screen until the physical end of frame. The MCU fetches the Status Row data then waits until its display is completed. It then performs the actions described below.

If LPEN has been enabled and a rising edge on the LPEN input has been detected, the LPENROW and LPENCOL positions in the command block are updated and the LPU status bit is set. If a Channel Attention has occurred, i.e., if CA has been activated, the command byte is fetched from command block and the specified channel command is executed. If the command issued is a "Stop Display" command, the MCU will terminate the display process and wait for the next channel attention. Otherwise, the MCU resumes the data access task by reinitializing pointers for the new frame and continues to fill the Row Buffers.

VEOF

EOF command in a virtual process datastream is interpreted as VEOF (Virtual End of Frame). It provides for reinitialization of LPTR using LIST-SWITCH, LBASE0 and LBASE1 for each process, analogous to the automatic reinitialization of LPTR at each end of frame in a Normal Display.

LPTR for the current process is reinitialized using LISTSWITCH, LBASE0 and LBASE1 contained in the process header. The End of Display (EOD) bit in the header is set to 1. The current process is terminated as in a VEOR and the next process in Access Table is accessed.

EOL

15	14	8	7	0
1	000	0010	XXXX	XXXX

The EOL (End of Line) command has a combined effect of NXTROW and NXTSTRG commands. All the actions performed in a END OF ROW command are carried out. In addition a END OF STRING command is executed before resuming the data access task. Thus, following the end of row, the data access is continued with the next data string. In virtual process datastream, this command has the combined effect of VEOR and END OF STRING.

FULROWDESCRPT

15	14	8	7	0
1	000	0011	n	

The next "n" words fetched from memory are loaded into the Row Characteristics holding registers. "n" is specified by the lower order byte of the command word and should be between 0 and 7.

The parameters loaded by this command will be used to define the row characteristics at the time the row currently being loaded is displayed. The data words defining these characteristics which follow the FULROWDESCRPT command must be ordered and organized in memory in a specifc format. The format for FULROWDESCRPT parameters is shown below in Figure 6e starting with "Lines Per Row" as the first parameter loaded.

This command will be ignored if encountered in a virtual process datastream. The MSB of all the parameters must be zero for proper operation in virtual display.

	Upper Byte								Lower Byte							
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Lines per row	—	—	—	—	RVV ROW	BLK ROW	DBL HGT	W DEF	—	—	—	LPR				
Normal Start/Stop	—	—	—			NRMSTRT			—	—	—	NRMSTOP				
Superscript Start/Stop	—	—	—			SUPSTRT			—	—	—	SUPSTOP				
Subscript Start/Stop	—	—	—			SUBSTRT			—	—	—	SUBSTOP				
Cursor 1 Start/Stop	—	—	—			CUR1 STRT			—	—	—	CUR1 STOP				
Cursor 2 Start/Stop	—	—	—			CUR2 STRT			—	—	—	CUR2 STOP				
Underline Line Selects	—	—	—			UL2 LINE SEL			—	—	—	UL1 LINE SEL				

RVV ROW, when this bit is set the CRVV pin will be inverted for the next full row.

BLK ROW, when this bit is set the row will be blanked (BLANK high).

DBLHGT, when the double height bit is set, all character are displayed with twice the scan lines per row.

WDEF, when the width defeat bit is set, the WDEF pin is activated for the entire row.

The following can be programmed from 0 to 31 yielding a range of 1 to 32 lines.

LPR specifies number of lines per row.

NRMSTRT, SUPSTRT, SUBSTRT specify line numbers in a display row which mark the start of normal, superscript and subscript characters respectively.

NRMSTOP, SUPSTOP, SUBSTOP specify line numbers in a row where normal, super script and subscript characters end respectively.

CUR1 STRT, CUR2 STRT specify the starting line numbers in a row for cursor 1 and cursor 2 respectively.

ULINE1 SEL, ULINE2 SEL specify the line numbers in a row where underline 2 will appear respectively.

All FULROWDESCRPT parameters affect the row in which they are programmed and stay in effect until changed by another FULROWDESCRPT command.

Figure 6e. Format for FULROWDESCRPT

SL SCROLL STRT

15	14	8	7	5	4	0
1	000	0100	XXX		SCR LINE	

The Slow Scan register in 82C3 is loaded with the scroll line specified by the five least significant bits of the command word. When the row currently being loaded is displayed, the line count for that row will start with the value specified by the Slow Scan register. A "Margin" (MGN) parameter, loaded by MODESET, specifies the number of blank lines plus one to be added at the top of the slow scroll field on the screen. This ensures the availability of sufficient DMA time for fetching the next row, when only a small number of scan lines are displayed in the top row of slow scroll window. This command is used for starting a slow scroll. (Note: MGN = 0 results in no margin buffer lines)

This command will be ignored if encountered in a virtual process datastream or if a SL SCROLL END command is encountered later on the same row.

SL SCROLL END

15	14	8	7	5	4	0
1	000	0101	XXX		END LINE	

The scroll location in row characteristics holding registers is loaded with the number of lines specified by the five least significant bits of the command word. This number specifies the number of lines to be displayed when the row currently being loaded is displayed. This is used instead of the regular LPR (Lines Per Row) characteristics, for this particular row. This command is used in the last row of a slow scroll for terminating a slow scroll. The Margin (MGN) parameter, loaded by MODESET, is used in the same way as in slow scroll start except that the specified number of blank lines are inserted at the bottom of the slow scroll in this case. This command will be ignored if encountered in a virtual process datastream or if followed by a SL SCROLL STRT on the same row.

TAB TO n

15	14	8	7	0
1	000	0110	"n"	

The lower byte of the command word specifies the column (RCLK count) after SYNCSTRT at which a Tab should occur. At display time, after the character preceding the Tab command is displayed, the screen is blanked until the RCLK count specified by the command ("n") is reached. After reaching the specified count, display is resumed by displaying the character following the TAB command.

If the RCLK count specified by the Tab command has already occurred before beginning the blanking for Tab, the display will be blanked until the end of the row.

This command is ignored, if encountered in a virtual display process datastream.

LD MAX DMA COUNT

15	14	8	7	0
1	000	0111		MAX COUNT

The Max Count register in 82730 is loaded with the Max DMA Count specified by the lower byte of the command word. The DMA Counter is also reinitialized with the Max Count value in the Command Block after all channel commands.

MAX DMA Count is programmable in the range of 1 to 256 (MAX COUNT value 0 equals 256). However, counts greater than the row buffer capacity will cause row buffer overruns if the data strings depend on MAX DMA to terminate the fetching.

The DMA counter is decremented for each data word as the Row Buffer is being loaded. Datastream commands and words supplying parameters for datastream commands as in FULROWDESCRPT, are not counted. Superscript/Subscript characters are counted in pairs, i.e., a pair of characters causes only one count.

In virtual screen display, every time a new process is accessed, the DMA counter is initialized with the Max DMA Count contained in the process header. This value is also stored in a Max Counter register.

At virtual end of row (VEOR) the Max Count register is written to the process header. The "LD Max DMA Count" command is ignored if encountered in a virtual process datastream.

ENDSTRG

15	14	8	7	0
1	000	1000	XXXX	XXXX

The SPTR register in the 82730 is loaded with a new String Pointer (SPTR) value fetched from the memory location indexed by the List Pointer (LPTR), which is stored in the LPTR register. The

LPTR register is incremented by two if a 16-bit addressing option is used and by four if 32-bit addressing is used. When more than one 82730 is connected in a cluster, each of them adds an offset, determined by its position in the cluster, to the pointer fetched from memory, before storing it in its SPTR register.

This command directs the data access to the next data string in the list of strings indexed by LPTR. The operation of this command is identical for a Virtual or Normal Display. In virtual display, the next data string within the current display process is accessed.

SKIP n

15	14	8	7	0
1	000	1001	n	

The next "n" data words fetched from memory are ignored. "n" is specified by the lower byte of the command word and is programmable from 0 to 255. If n equal to 0 is specified, no words are skipped. Any datastream commands encountered in the data fetch are not counted towards these n words. Also parameters following the datastream command as in FULROWDESCRPT are not counted. All embedded datastream commands are executed with the following exceptions.

If a Tab To N data stream command is encountered during the execution of a Skip N command, the Tab command will result in a NOP, i.e. a Tab embedded in the data to be skipped will be ignored.

If an EOL (End Of Line) data stream command is encountered during the execution of a Skip N command, it will be executed with the following effect. In non-auto line feed mode, (ALF = 0) the EOL command has the combined effect of End Of Row and End Of String commands. In auto line feed mode, (ALF = 1) the EOL command has the effect of an End Of String command only.

If the data words skipped include any superscript-subscript characters, they are skipped in pairs and a pair of characters is counted as only one count in "n". If another skip command is encountered its value of "n" is added to the present skip count and skipping continues.

REPEAT n

15	14	8	7	0
1	000	1010	n	

The next data word (byte, if DTW16=0) fetched from memory is stored in the Row Buffer "n" times, where "n" is specified by the lower byte of the command word. "n" is programmable from 0 to 255. If n equal to 0 is specified no repetitions will occur, and the word following the Repeat n command will be ignored. This character will eventually be displayed n times. The DMA counter is also made to count n times. In non-auto linefeed mode (ALF = 0), reaching Max DMA Count before the n repetitions are completed will result in a termination of the Repeat n command. This command will also be terminated if the Row Buffer gets filled completely before the n repetitions are completed.

It should be noted that the data word immediately following the Repeat n command is treated as character data, irrespective of the value of its command bit.

SUP/SUB n

15	14	8	7	0
1	000	1011	n	

The next "n" pairs of data words (bytes, if DTW16 = 0) fetched from memory are treated as superscripts or subscript characters. "n" is specified by the lower byte of the command word. These n pairs are assumed to be ordered with the superscript preceding the subscript.

No datastream commands are permitted in the 2n words following this command. All of these words are interpreted as superscript-subscript pairs. The DMA counter is made to count only once for each pair of characters. In non-auto linefeed mode (ALF=0), reaching the Max DMA Count will result in a termination of this command. If n equal to zero is specified, no action will result.

RPT SUB/SUP n

15	14	8	7	0
1	000	1100	n	

The operation of this command is similar to that of the "Repeat n" command except that the pair of characters following the "RPT SUB/SUP n" command is repeated n times. "n" is specified by the lower byte of the command word and is programmable from 0 to 255. If n equal to zero is specified, no repetitions will occur, and the two data words following the "RPT Sub/Sup n" command will be ignored. The two data words (bytes, if DTW16=0) immediately following the command word are interpreted as a superscript-subscript pair and are repeated. The DMA counter is made to count only once for each repetition of the pair. In non-auto linefeed mode (ALF=0), reaching Max DMA Count prior to completion of n repetitions will cause a termination of this command.

SET GEN PUR ATTRIB

15	14	8	7	0
1	000	1101	GPA OPERAND	

This command provides control over the output pins assigned to General Purpose Attributes, GPA1 through GPA4.

	7	6	5	4	3	2	1	0
GPA OPERAND	GPA4 DATA	GPA4 EN	GPA3 DATA	GPA3 EN	GPA2 DATA	GPA2 EN	GPA1 DATA	GPA1 EN

ENCODING

GPAx DATA	GPAx EN	FUNCTION
0	0	ROW BUFFER DATA
1	0	ROW BUFFER DATA
0	1	GPA DATA = 0
1	1	GPA DATA = 1

The GPA in the Process Header is updated each time a SET GPA command is executed. Thus the GPA state in the header is updated to reflect any changes caused by the "Set Gen Pur Attrib" command. The GPA command occupies a character space on the screen. Consequently, a GPA command is counted as a character towards MAX DMA count. However, a GPA command nested in a Skip N or a TAB to N command is skipped, i.e., it has no effect.

The encoding of the operand, specifying GPA operation, is shown below.

SET FIELD ATTRIB

15	14	8	7	0
1	000	1110	XXXX	XXXX
0	FIELD ATTRIBUTE MASK			

The word following this command is fetched. This word is used as a Field Attribute Mask in storing all subsequent display data words in row buffer. The bits in the data words fetched from memory corresponding to the bit positions containing a "1" in Field Attribute Mask are all set to 1 before storing the data word in row buffer. The Field Attribute Mask is used on all display data words fetched from memory. The mask register will contain all 0's upon reset and is cleared at the beginning of each frame.

NOP

15	14	8	7	0
1	1XX	XXXX	XXXX	XXXX

No action is taken. The data access task is resumed by fetching the next data word.

Datastream Command Conventions

The reaching of Max DMA Count, encountering of terminating commands such as ENDROW, EOF, etc. and occurrences of these while executing a "skip n" command give rise to various possible combinations of events. The behaviour of 82730 under these circumstances is described below:

i) When Max DMA Count is reached, it has the effect of a VEOR command if a Virtual Display is in progress or a ENDROW command if a Normal Display is in progress. It also causes an automatic end of string i.e., the effect of a NXTSTRG command in non-auto linefeed mode (ALF = 0).

.i) In non-auto linefeed mode, "Repeat n", "Sub/Sup n" and Rpt Sub/Sup n" commands are terminated upon reaching a max DMA count, even if "n" is not reached.

iii) "Skip n" command is terminated if EOF command is encountered. It is also terminated upon encountering a ENDROW command in non-auto linefeed mode (ALF = 0).

iv) "Repeat n" "Sub/Sup n" and "RPT Sub/Sup n" commands can be nested within a "Skip n" command. If superscript-subscript characters are skipped, each pair of characters counts as one skipped character. If the above commands are encountered during a "skip n" and if the specified count (n) in these commands is not reached by the end of execution of the "skip n" command, the execution of the nested command is continued beyond the termination of "skip n" command until the remaining portion of the count specified in the nested command is completed.

ORDER NUMBER: 210931-004

VIRTUAL SCREEN MODE

Command Process Commands

In Virtual Screen Display, 82730 accesses display processes and command processes through the Access table. The command processes enable the I/O Driver process to direct 82730 to execute certain data stream commands by inserting an appropriate command process address in the Access table. This capability enables the preservation of uniformity and consistency of operation between normal and virtual environments, by assigning different interpretations to the command according to the access environment. It is especially useful for termination and initialization commands. The operation of command process commands is analogous to that of data stream commands except for a different access environment.

Command Process Command List

The commands allowed in command processes can be divided into two subsets. The first subset consists of commands that can be issued only through a command process, while the second one consists of normal datastream commands that can also be issued through a command process. The command code for a datastream command issued through a command process is the same as that for the normal datastream command embedded in the data. However, certain datastream commands are interpreted differently when they are issued through a command process as opposed to embedding in the datastream of a virtual display process. The most significant bit (MSB) of the command word must be a "1". In the datastream, this bit distinguishes a command word from character data. In the process environment, this bit distinguishes a command process from a display process. The commands permitted in command processes are listed below. No other commands will be recognized if encountered in a command process and will result in a NOP. All undefined command codes apart from those designated as NOP are reserved and should not be used. Encountering an illegal command code causes the RDC (Reserved Datastream Command) status bit to be set and will generate an interrupt, if enabled.

Table 5. Command Process Command List

COMMAND	INTERPRETATION IN VIRTUAL PROCESS DATASTREAM	COMMAND CODE		OP CODE
		OP CODE	PARAMETERS	
Command Process Only Command:				
1 INIT NEXT PROCESS	NOP	1000 1111	XXXX XXXX	8F
Command Process or Datastream Commands:				
2 ENDROW	VEOR	1000 1000	XXXX XXXX	80
3 EOF	VEOR	1000 0001	XXXX XXXX	81
4 EOL	VEOR + NXTSTRG	1000 0010	XXXX XXXX	82
5 FULROWDESCRPT	NOP	1000 0011	"n"	83
6 SL SCROLL STRT	NOP	1000 0100	XXX "SCR LINE"	84
7 SL SCROLL END	NOP	1000 0101	XXX "END LINE"	85
8 TAB TO n	NOP	1000 0110	"n"	86
9 LD MAX DMA COUNT	NOP	1000 0111	"COUNT"	87
10 (RESERVED)	RESERVED	10XX XXXX	XXXX XXXX	90-BF
11 NOP	NOP	11XX XXXX	XXXX XXXX	C0-FF

210931-004

INIT NEXT PROCESS

15	14	8	7	0
1	000	1111	XXXX	XXXX

This command can be used only in a command process to initiate a virtual display "window".

Upon receiving this command, the command process is terminated and the next process in Access Table is accessed by fetching the new process address. However, the LPTR register is not directly loaded from the LPTR location in the process header. Instead, LISTSWITCH in the process header is examined and LPTR is initialized with the value LBASE 0 or LBASE 1 depending upon whether LISTSWITCH is 0 or 1 respectively. Both LBASE0 and LBASE1 are contained in the header.

The process header format is shown in Figure 7. Also the End of Display Bit (EOD) in the header is reset.

The data access task for a virtual display is then resumed, with this value of LPTR.

	15	14	13	8	7	6	0	LOCATION
LS: LISTSWITCH ALF: AUTO LINE FEED	0	----			EOD		----	PROCESS ADDR
		----			LS ALF			PROC ADDR + 2
		----					MAX DMA COUNT	PROC ADDR + 4
				LBASE0 LOWER				PROC ADDR + 6
				LBASE0 UPPER				PROC ADDR + 8
				LBASE1 LOWER				PROC ADDR + 10
				LBASE1 UPPER				PROC ADDR + 12
	1	----					GPA	PROC ADDR + 14
	1				FIELD ATTRIBUTE MASK			PROC ADDR + 16
				LPTR	LOWER			PROC ADDR + 18
				LPTR	UPPER			PROC ADDR + 20
SAVE AREA				SPTR	LOWER			PROC ADDR + 22
				SPTR	UPPER			PROC ADDR + 24
	RPT S/S	S/S	RPT	— —		REPT COUNT		PROC ADDR + 26
	1			REPT CHAR				PROC ADDR + 28
	1			REPT CHAR 2				PROC ADDR + 30

PROCESS ADDR

15	14	8	7	0
1		COMMAND		

C/$\overline{D}$

Figure 7. Process Header for Display and Command Process

210931-004

ENDROW

15	14	8	7	0
1	000	0000	XXXX	XXXX

The actions performed by a ENDROW datastream command in a Normal Display are carried out. The next process in Access Table is accessed and the data access task is resumed, after the next Row Buffer swap

EOF

15	14	8	7	0
1	000	0001	XXXX	XXXX

The actions performed by an EOF (End of Frame) data stream command in a Normal Display are carried out.

EOL

15	14	8	7	0
1	000	0010	XXXX	XXXX

This command is identical to ENDROW command in Virtual Display in Command Process environment. ENDSTRG, which is strictly a data operation within a display process is meaningless in the command process environment.

FULROWDESCRPT

15	14	8	7	0
1	000	0011	"n"	

The actions performed by the FULROWDESCRPT datastream command are carried out. The data access task is resumed by accessing the next process in the Access Table.

SL SCROLL STRT

15	14	8	7	5	4	0
1	000	0100	XXX		"SCR LINE"	

The same actions as the SL SCROLL STRT datastream command. The data access is resumed with the next process in Access Table.

SL SCROLL END

15	14	8	7	5	4	0
1	000	0101	XXX		"END LINE"	

The actions performed by a SL SCROLL END datastream command, in a Normal display, are carried out. The data access task is resumed with the next process in Access Table.

TAB TO n

15	14	8	7	0
1	000	0110	"n"	

The effect of this command process command is identical to that of the TAB TO n datastream command. The TAB can be used to establish the left edge of a virtual display "window".

LD MAX DMA COUNT

15	14	8	7	0
1	000	0111		MAX COUNT

The Max Count register on 82730 is loaded with the value specified by the lower byte of the command word. The DMA counter is also initialized with this Max Count Value.

The next process in the Access Table is accessed. However, the Max DMA Count value in the process header is not used for initializing the DMA counter. Instead, the DMA counter as initialized by the LD Max DMA Count command is used for this process. The virtual display data access task is then resumed normally. When the process is terminated, the new Max Count value is written to the process header. Thus the Max Count value in the header is updated as a result of this command.

NOP

15	14	8	7	0
1	1XX	XXXX	XXXX	XXXX

No action is taken. Data access task is resumed by fetching the next process address from Access Table.

ERROR AND STATUS HANDLING

Error Conditions

Since the MCU and DG function asynchronously with respect to each other, different relative timings in MCU and DG operation are possible, some of which result in error conditions. The lack of appropriate termination commands for row or frame data in the datastream also gives rise to certain error conditions. These types of situations occurring in display process operation are described below.

In normal operation, DG initiates a buffer swap at the physical end of a display row. If the MCU has not finished loading its row buffer by that time, a "Data Underrun" occurs. This results in

blanking of the screen until physical end of frame by DG and execution of an EOF (End of Frame) command by MCU. Data underrun also occurs when the first row of the frame has not finished loading by the start of the character field. The entire frame will be blanked in this case.

If a physical end of frame is reached prior to encountering an EOF datastream command, a "Frame Data Error" occurs, which results in the execution of an EOF command by MCU. (Note that this does not disrupt the visible display action, and may not constitute an error for certain data structures. The error indication is included as a flag where knowledge of this condition is desired.) Similarly, when the MCU fills up a row buffer completely, without encountering a END-ROW command, the "Data Buffer Overrun" flag is set.

All of the above conditions result in the setting of an appropriate status bit and generation of an interrupt if the corresponding interrupt has been enabled.

VDIP: Virtual Display In Progress
DIP: Display In Progress
RCC: Reserved Channel Command
RDC: Reserved Datastream Command
FDE: Frame Data Error

DUR: Data Under Run

This status bit is set by Display Generator if the Microcontroller Unit (MCU) has not finished loading its Row Buffer when the DG initiates a buffer swap at the physical end of a display row. This condition is defined as data underrun and causes the MCU to execute an EOF command and the DG to blank the screen until the physical end of frame.

LPU: Light Pen Update

This status bit is set by the MCU after updating the LPENROW and LPENCOL locations in command block. The detection of LPEN input is enabled by the LPEN ENABLE channel command. The detection of a rising edge on the LPEN input causes the current row and column position to be stored internally. The MCU updates the LPEN ROW and LPEN COL locations in command block at the next end of frame and sets the LPU status bit. Further updates of these command block locations are inhibited until another LPEN ENABLE command is issued.

15	9	8	7	6	5	4	3	2	1	0
(RESERVED)		VDIP	DIP	RDC	RCC	FDE	EOF	DBOR	LPU	DUR

Status and Interrupt Handling

A status word is maintained in an internal register by 82730 and it is written to the "STATUS" location in command block when the "Read Status" channel command is executed. The processor can thus read status information by issuing this command. the processor can also enable interrupts for certain status bits by specifying an interrupt mask which is loaded in 82730 as a result of a "Load Int Mask" channel command. This establishes a communication mechanism between 82730 and the processor for error and status reporting.

Status Word

The format for the status word is shown below. The function of each of the status bits is described below.

The status bits get set under the conditions described above. Interrupts can be enabled for all status bits except DIP and VDIP bits. The interrupt status bits are cleared at the beginning of each new display field. DIP and VDIP bits are cleared only after receiving a "STOP DISPLAY" command or a Reset.

All status bits are cleared by a Reset.

EOF: End of Frame
DBOR: End of Row
LPU: Light Pen Update
DUR: Data Under Run

DBOR: Data Buffer Over Run

This status bit is set when the MCU tries to fill a row buffer beyond its capacity. The MCU will stop fetching characters after this point and the display is blanked following the completion of the row currently being displayed.

EOF: End of Frame

This bit is set by the DG at the physical end of the nth frame, where 'n' is specified by the MODESET parameter FRAME INTERRUPT COUNT. This provides the means for timing frame related events such as slow scrolls.

FDE: Frame Data Error

This status bit is set by the DG at the physical end of frame if no EOS datastream command has been encountered until then. This also results in the execution of the EOS command by the MCU.

RCC: Reserved Channel Command

This bit is set by the MCU upon encountering an illegal datastream or command process command. This can be used to trap software errors during program development.

RDC: Reserved Datastream Command

This bit is set by the MCU upon encountering an illegal datastream or command process command. This can be used to trap software errors during program development.

DIP: Display In Progress

This bit is set by the MCU immediately after receiving a "Start Display" channel command. It remains set as long as the display process is active and is reset upon receiving a "Start Virtual Display" or "Stop Display" command or a Reset. Interrupts cannot be enabled for this status bit.

VDIP: Virtual Display In Progress

This bit is set by the MCU immediately after receiving a "Start Virtual Display" channel command and is reset upon receiving a "Start Display" or "Stop Display" command or a Reset. This bit remains active as long as the virtual display process is active. Interrupts cannot be enabled for this status bit.

Interrupt Processing

The system processor can enable interrupts on any of the status bits, with the exception of DIP and VDIP bits, by specifying an interrupt mask. A "1" in a bit position in the interrupt mask disables (masks out) interrupts on the status bit located in the corresponding bit position in the status word. The format for Interrupt Mask is shown below. The Int Mask can be loaded into 82730 from the INTMASK location in command block by a "Load Int Mask" channel command.

If the interrupt is enabled for a particular status bit by programming a "0" in the corresponding bit position in INTMASK and if the status bit gets set during the course of the display, an interrupt will be generated by 82730 at the next end of frame. At the end of frame, the 82730 will first perform the tasks of updating LPEN position (if required) and servicing the Channel Attention (if CA was activated). Then the status word in the internal register will be written to the INT GENERATION CODE location in the Command Block and the SINT output will be activated. The SINT pin is not deactivated until an interrupt reset signal is received at the IRST pin.

82730 continues to perform its normal display task after activating the SINT pin. If no interrupt reset is received until the next end of frame then any new interrupts that might have been generated at that end of frame will be lost. Therefore, it is essential for the system processor to issue an interrupt reset within a frame time after an interrupt is generated.

When the display is not activated, the only interrupt that can occur is the Reserved Channel Command interrupt. Upon receiving an invalid channel command, 82730 will write the status word to INT Generation Code location in the Command Block and activate SINT output, if that interrupt is enabled.

The processor can use the interrupt capability to get status information from 82730. A possible interrupt service routine for the system processor is shown in flow chart form in Figure 9.

15 7	6	5	4	3	2	1	0
(RESERVED)	RDC INT MASK	RCC INT MASK	FDE INT MASK	EOF INT MASK	DBOR INT MASK	LPU INT MASK	DUR INT MASK

INT MASK = 0 Enables the corresponding interrupt.
INT MASK = 1 Masks or disables the corresponding interrupt.

Figure 8. Interrupt Mask

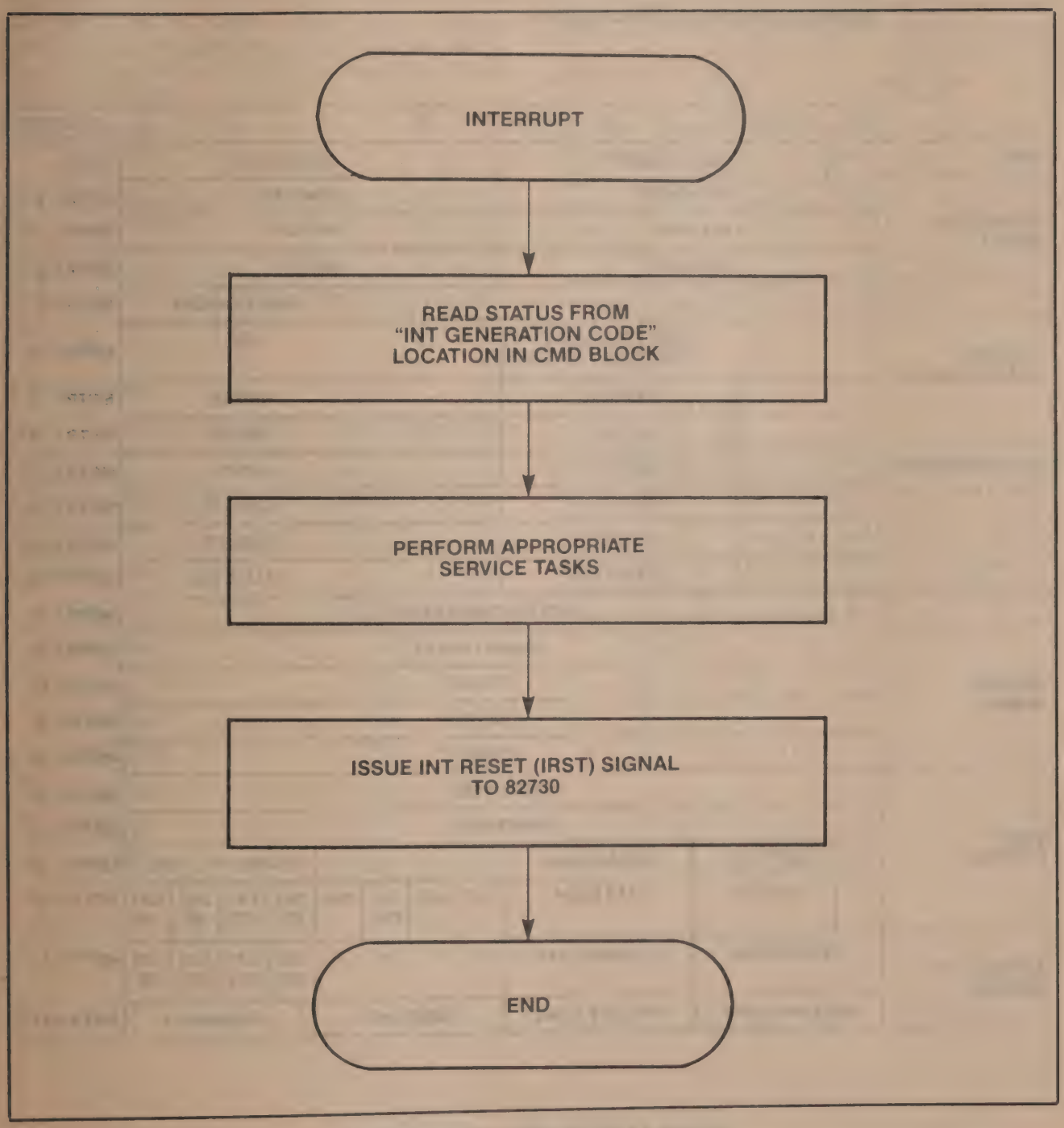


Figure 9. Interrupt Service Routine For System Processor

82730 VIDEO INTERFACE

The Mode Pointer in the Command Block points to a parameter block containing the Mode information required for the display. The organization of the mode words in the Mode Block is shown below.

FROM MODE POINTER IN COMMAND BLOCK

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	LOCATION
DMA	-	BURST LENGTH							-	BURST SPACE							MPTR
HORIZONTAL MODES	LINE LENGTH								HSYNCSTP								MPTR = 2
	HFLDSTRT								HFLDSTP								MPTR = 4
	HBRDSTRT								HBRDSTP								MPTR = 6
	-	-	-	-	-	-	-	-	-	-	-	SCROLL MARGIN					MPTR = 8
CHAR ROW CHARACTERISTICS	-	-	-	-	RVV ROW	BLK ROW	DBL HGT	W DEF	-	-	-	LPR					MPTR = 10
	-	-	-	NRMSTRT					-	-	-	NRMSTP					MPTR = 12
	-	-	-	SUPSTRT					-	-	-	SUPSTP					MPTR = 14
(FULROWDESCRPT)	-	-	-	SUBSTRT					-	-	-	SUBSTP					MPTR = 16
	-	-	-	CUR1STRT					-	-	-	CUR1STP					MPTR = 18
	-	-	-	CUR2STRT					-	-	-	CUR2STP					MPTR = 20
	-	-	-	U2 LINE SEL					-	-	-	U1 LINE SEL					MPTR = 22
	-	FIELD ATTRIBUTE MASK															MPTR = 24
VERTICAL MODES	-	-	-	-	-	FRAME LENGTH											MPTR = 26
	-	-	-	-	-	VSYNCSTP											MPTR = 28
	-	-	-	-	-	VFLDSTRT											MPTR = 30
	-	-	-	-	-	VFLDSTP											MPTR = 32
BLINK CONTROL	(RESERVED)																MPTR = 34
	(RESERVED)																MPTR = 36
	-	DUTY CYC		CURSOR BLINK					-	-	-	-	FRAME INT COUNT				MPTR = 38
	-	DUTY CYC		CHAR BLINK					ILE	RFE	B POL	BUE	CR2 CD	CR1 CD	CR2 BE	CR1 BE	MPTR = 40
ATTRIBUTE BIT SELECTS	REVERSE VIDEO			BLINKING CHAR					-	-	-	-	CR2 RVV	CR1 RVV	CR2 OE	CR1 OE	MPTR = 42
	ABS LINE COUNT			INVISIBLE CHAR					UNDERLINE 2				UNDERLINE 1				MPTR = 44

Figure 10. Mode Block Organization

210931-004

CAM ARRAYS

Three Content Addressable Memory arrays are used for generating timing parameters to control the video display: the HORIZ MODE CAM, the VERT MODE CAM andthe CHAR ROW CAM. The user has the flexibility to define his own timing parameters by loading them into the CAM arrays via the MIU. All of these parameters can be modified at the end of every frame. All the parameters in the CHAR ROW CAM, except MARGIN, are changeable on a row by row basis. Each of the three CAM arrays is described separately below:

Timing Sources

RCLK and CCLK inputs are provided by the external video logic to the 82730. The RCLK is used to increment the HORIZ COL CNTR and hence generates all horizontal timing parameters. CCLK is used to clock the character and attribute data output from the 82730 to the external display dot logic. Data changes on the positive going edge of RCLK or CCLK.

Initialization

Upon activation of the RESET input, the 82730 display generator will stop all operations in progress and deactivate all outputs. It will stay in this quiscent state until the MIU executes the MODESET command. The following table shows the states of all the Display Generator outputs during and after RESET.

Pin Name	Condition
DAT0-14	Low
WDEF	Low
LC0-4	High
BLANK	Low
CSYNC	High
$\overline{\text{CHOLD}}$	High
HSYNC	Low
VSYNC	Low
CRVV	Low
RRVV	Low

After reset of the 82730, the CAM arrays are in undetermined states. The CAM arrays are set upon the execution by the MIU of the MODESET command. The HORIZ and VERT MODE CAM contents are especially critical since they are used to generate timing control signals to the external video logic. Without the generation of the timing signals, no display process can take place. Hence, START DISPLAY command cannot be executed before the first MODESET command after the device reset. The START DISPLAY command will be ignored if it precedes the MODESET command.

The row buffers also contain unknown information after power up and reset. In executing the START DISPLAY command, the MIU would first load the two row buffers with the first two rows of character data to be displayed. Upon completion of loading of both buffers, it will signal the DG to begin the display process. In this way, only valid character data will be output to the external video logic.

Timing Parameters

The timing parameters read from the MODESET Block and stored in the VERT MODE CAM and HORIZ MODE CAM are used to control the video display and they can be best illustrated in the Map of Timing Parameters shown below. All of these timings have to be defined after power up and reset and can be changed on a frame by frame basis during display.

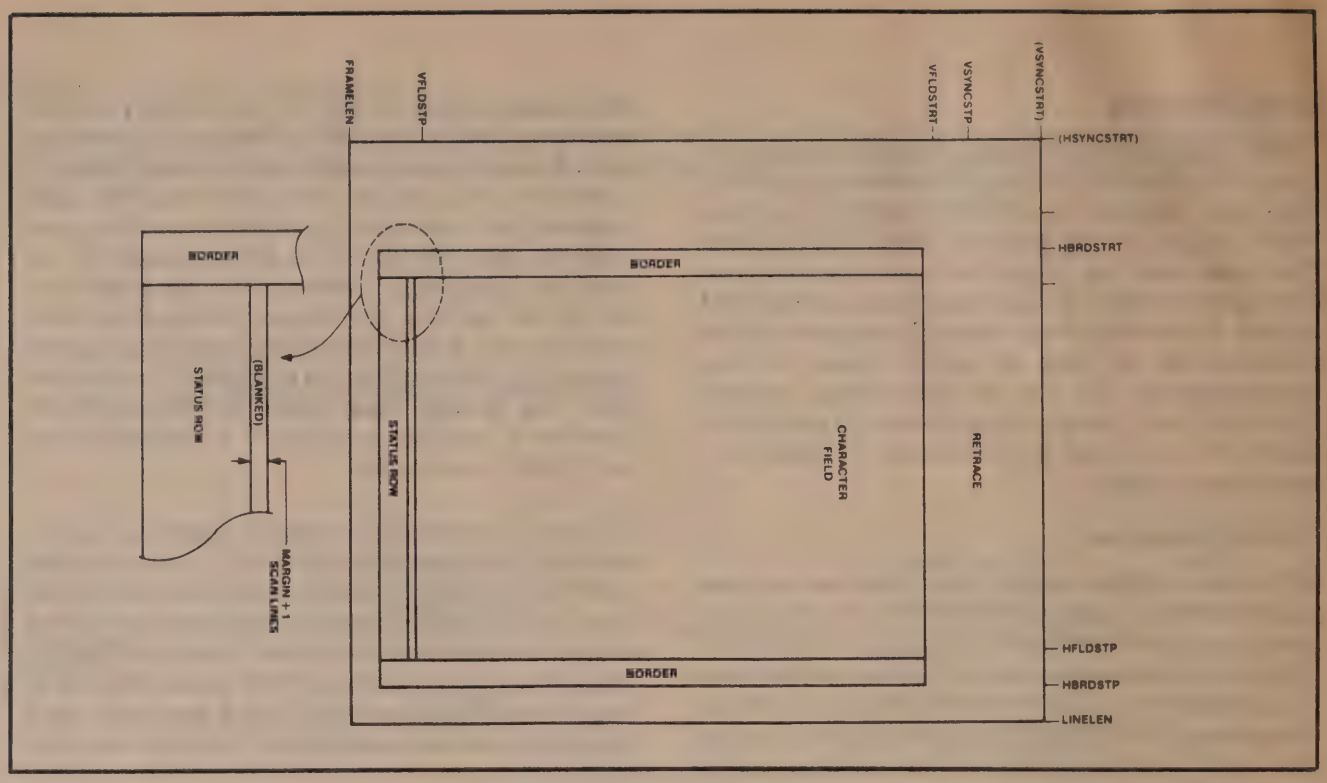


Figure 11. Timing Parameters

Row Timing Parameters

The row timing parameters are stored in HORIZ MODE CAM and are programmable from 0 to 255 RCLK times. These parameters are:

(a) HSYNCSTRT - Horizontal Sync Start. The RCLK count on each scan line where HSYNC pin is activated. This parameter is not programmable. The RCLK period that follows the rising HSYNC edge is defined as column zero. It is used as the reference for all other horizontal timing parameters.

(b) HSYNCSTP - Horizontal Sync Stop. The RCLK count on each scan line where the HSYNC pin is deactivated. The falling edge of HSYNC occurs at the leading edge of the programmed RCLK period.

(c) LINELEN - Line Length. This parameter defines the total number of RCLK's in each scan line including display time, border and horizontal retrace time. There are LINELEN + 1 RCLK periods per horizontal line scan.

(d) HBDRSTRT - Horizontal border start. The RCLK count on a scan line where the border begins. The border begins at the leading edge of the programmed RCLK period.

(e) HBDRSTP - Horizontal Border Stop. The RCLK count on a scan line where the border ends. The border terminates at the leading edge of the programmed RCLK period.

(f) HFLDSTRT - Horizontal Field Start. The RCLK count on a scan line where the character display field begins. If the row buffer is ready to be displayed, the CSYN pin will be deactivated at this point. This field begins at the leading edge of the programmed RCLK period.

(g) HFLDSTP - Horizontal Field Stop. The RCLK count on a line where the character display field stops. When this timing point is reached, CSYN will be activated. This field ends at the leading edge of the programmed RCLK period.

There is also one pseudo parameter, SYNCDLY. It is fixed at one half LINELEN and is used as the start and end timing for VSYNC in odd frames in interlaced displays. VSYNC starts at HSYNCSTRT in even frames for interlaced displays and all frames for non-interlaced displays.

210931-004

There are certain restrictions in the programming of HFLDSTRT and HFLDSTP and those restrictions are best illustrated below. There has to be at least 4 RCLKS in between HFLDSTRT and HFLDSTP of the same scan line and 15 RCLKS in between HFLDSTP of one line and HFLDSTRT of the next. The minimum delay of 15 RCLKS is for the charging of the pipeline from the row buffer to the character data output DAT0-DAT14 as well as the setting of the correct value for the scan line output LC0-LC4.

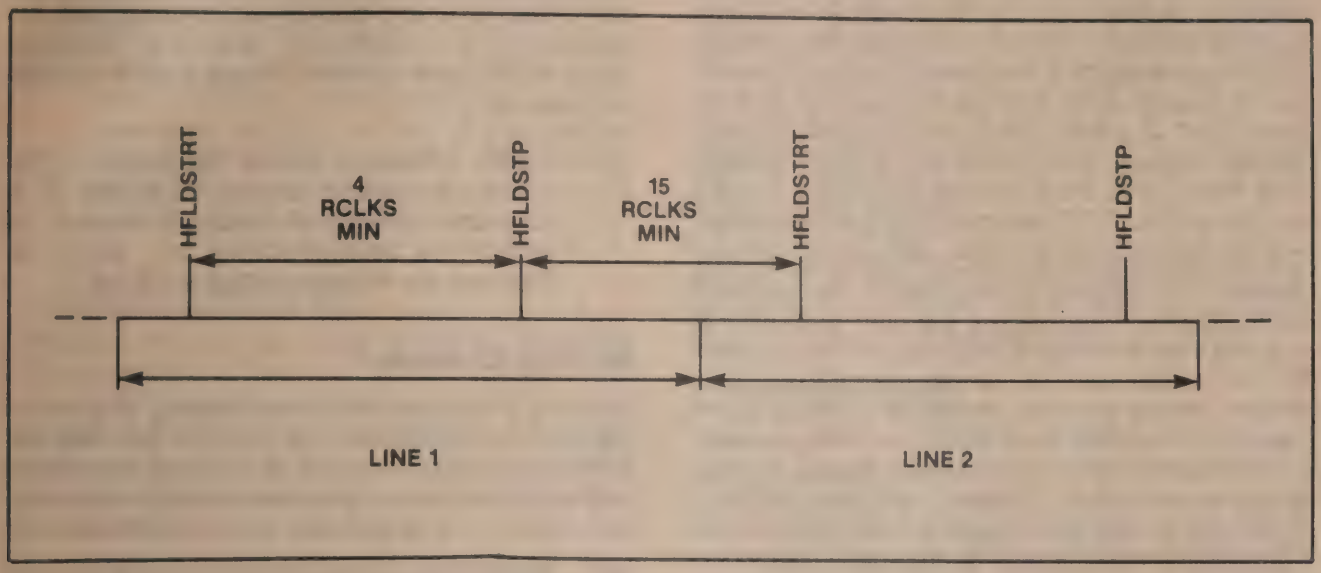


Figure 12. Horizontal Timing Restrictions

Frame Timing Parameters

Frame timing parameters are stored in the VERT MODE CAM and are programmable from 0-2047 scan lines. These parameters are:

(a) VSYNCSTRT - Vertical Sync Start. The line count where the VSYNC is activated. This occurs at the end of a field automatically. This parameter is not programmable. The rising edge of VSYNC occurs with the rising edge of HSYNC for all non-interlace fields and for odd fields in the interlace mode.

(b) VSYNCSTP - Vertical Sync Stop. The line count at which the VSYNC pin is normally deactivated. VSYNC changes at the rising edge of HSYNC normally. However it occurs at SYNCDLY at the beginning of odd fields of an interlaced display.

(c) FRAMELEN - Frame Length. This parameter defines the total number of scan lines per frame. It is used to reset the FRAME LINE CNTR. In an interlaced display, FRAMELEN must be an even number. If an odd number is programmed, one additional line will occur automatically. There will be FRAMELEN + 1 scan lines per frame. (Note that interlace mode contains two fields per frame).

(d) VFLDSTRT - Vertical Field Start. Programs the scan line count where the character display field begins.

(e) VFLDSTP - Vertical Field Stop. Programs the scan line count where the regular character display field ends. VFLDSTP times the beginning of the Status Row. The channel attention sequences, interrupt handling, row buffer swap and intialization for the next frame are started after the display of the Status Row is completed. See * below.

* (Character Field Boundry definition: The starting or ending event is defined to occur at HFLDSTP on the scan line following the programmed value. Thus the visible character field effectively begins two scan lines below the programmed start value and ends one scan line below the programmed stop value.)

Status Row

The Vertical Frame Timing Parameters have no border controls, unlike the Horizontal Row Timing Parameters. The top and bottom borders can be replaced with regular display rows that are video-reversed and contain no data. The top border is easily timed from VFLDSTRT. The bottom border is more difficult without help from the Vertical Timing generators. If there were no help, the user would have to keep track of the number of scan lines used in each row to know when to stop regular display and create the bottom border. This would also preclude his ending his regular display with an EOF command before the border.

The 82730 provides this help with the Status Row feature. The display of the Status row is timed from VFLDSTP and allows the user to display a row in a fixed position at the bottom of the screen that is independent of the regular data and any display errors (display ended by an EOF command or the DURN, DBOR, or FDE errors). (There is one dependency on the regular display data: the row format. The last FULROWDESCRPT (FRD) set in the regular data will be used on the Status Row unless a new command is issued for the row. It is recommended that the user include a new FRD command in the Status Row data to eliminate this dependency).

Status Row display starts SCROLL MARGIN plus one scan line after VFLDSTP. This margin is provided to insure enough DMA time if the regular display runs up to VFLDSTP. The user can create a bottom border or any end-of-display row that he chooses. A display status or system status line, or special programmable key function definition line can be implemented with this feature.

CHARACTER ATTRIBUTES

The 15 bits of the character word can be partitioned into character address and attribute bits.

Some common attributes may be individually defined and enabled or disabled by fields in the attribute parameter registers. Each attribute has two means of being enabled. The enable bits defined below are set during the MODESET channel command and are used as a global enable. The user does not have to enable the provided attributes. He may free more data bits for his own use this way. The second enable bit is contained in each character loaded to the row buffer to enable the attribute on a character by character basis. They are individually described in detail in the following sections.

Reverse Video

When a character with the reverse video attribute is displayed, the CRVV pin will be inverted during the time the character is being displayed. The reverse video affects the entire height of the row for that character space. For superscript/subscript pairs, the reverse video effect is controlled by superscript until SUBSTRT when the subscript attribute bit takes control. The parameter for this attribute is:

RVBS - Reverse Video Bit Select. This parameter selects one of the 15 bits of a character data word. Values 0 through 14 select the corresponding bit. Value 15 disables the Reverse Video attribute.

Blinking Character

When a character with the blinking character attribute is displayed, the BLANK pin will be activated and deactivated during the character display time according to programmable rate and duty cycle. The parameters for this attribute are:

(a) BCBS - Blinking Character Bit Select. Selects one of the 15 bits of a character data word as the blinking character attribute control. As with Reverse Video above, the value of the select determines the controlling bit or disables the attribute.

(b) CHAR BLNK FREQ - Selects one of the 32 blinking frequencies available for the blinking character and blinking underline. The character blink rate is calculated as below:

$$\text{Blink Rate} = \frac{\text{Frame Refresh Rate}}{4 \times \text{CHAR BLNK FREQ}}$$

(c) CHAR DUTY CYCLE - A 2-bit register to select 4 duty cycles available for blinking character and blinking underline. The selection logic is defined to be as follows:

00=100% always on
11= 75% on
10= 50% on
01= 25% on

Underline #1

When a character with underline is displayed, the BLANK Pin will be activated and the CRVV pin will be inverted during the time the scan line specified

by the underline select register is displayed. The parameters used to define underline #1 are:

(a) ULS1 - Underline Line Select 1. It determines which scan line of a character row will be used for the underline #1. This parameter is modifiable on a row by row basis by the FULROWDESCRPT command.

(b) ULBS1 - Underline Bit Select 1. This parameter can only be changed by MODESET. It selects one of the 15 bits of a character data word as the underline #1 attribute control. Again, a value of 15 in the select field disables this attribute.

Underline #2 (Blinking)

Underline #2 can be made to blink. When its blinking feature is deactivated, its visual effect is exactly the same as underline #1. When it is enabled to blink, its blink rate and blinking duty cycle are the same as those defined for blinking character. The parameters used to define this attribute are:

(a) UL2SEL - Underline Line Select 2. This parameter determines which scan line of a character will be the 2nd underline. It is changeable on a row by row basis by the FULROWDESCRPT command.

The next two parameters can only be modified by the MODESET Command.

(b) ULBS2 - Underline Bit Select 2. Selects one of the 15 bits of a character data word or GPA1 as the second underline attribute control. A bit select value of 15 disables the second underline.

(c) BUE - Blinking Underline Enable. Activation of this bit will cause the second underline attribute to start blinking.

Invisible

A character with this attribute will occupy its character position on the screen but will not be displayed (i.e. BLANK will be active). This attribute does not affect the Reverse Video attribute if they are programmed together. The parameter that is used to implement this attributes:

IBS - Invisible Bit Select. Selects one of the 15 bits of a character data word as the invisible attribute control. Value 15 disables the invisible attribute.

Absolute Line Cntr Attribute

This character attribute allows the display of special graphic characters, or may be used to upshift normal characters to implement displays with overlapping superscript and subscript fields. When a character with this character attribute enabled is being displayed, its LC0-LC4 pins will reflect the output from the CHAR ROW LNE CNTR which counts the absolute line count of a row. The activation of this attribute overrides the line count mode of both normal and subscript/superscript characters. The parameter used to select the attribute is:

ABS LINE BIT SEL. This four bit register selects one of the 15 bits of a character data word as the absolute line counter output attribute control. Select value 15 disables the ABS Line attribute.

Cursor Generation

The cursor characteristic parameters are changeable on a frame by frame basis by MODESET.

(a) CUR FREQ - Cursor frequency. Selects the blinking frequency for both cursors. The selection logic is similar to CHAR BLNK FREQ

(b) CUR DUTY CYCLE - Cursor duty cycle. Selects the blinking duty cycle for both cursors. Its selection logic is similar to CHAR DUTY CYCLE.

(c) CR1RVV - Cursor 1 Reverse Video Enable selects a reverse video type cursor as opposed to a solid (blanking) cursor.

(d) CR1BE - Cursor 1 Blink Enable changes the cursor 1 block or underline to a blinking block or underline. Enabling this bit also causes DAT 14 pin to "blink" as well, if the CR10E bit is set.

(e) CR10E - Cursor 1 Output Enable reconfigures the DAT 14 pin to indicate when cursor 1 is active. CR20E enabled directs the cursor 2 signal to DAT 13 pin in a similar fashion.

(f) CR1CD - Cursor 1 Light Pen Cursor Detect directs the CCLK cursor #1 position to be translated to its nearest equivalent RCLK position through the LPEN facility.

An identical set of parameters (c) through (f) is available for the generation of CURSOR 2. The two cursors share the same FREQ and DUTY CYCLE parameters.

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature under Bias .... 0°C to 70°C
Storage Temperature ..... −65°C to +150°C
Voltage on Any Pin with
Respect to Ground ..... −1.0V to +7V
Power Dissipation ..... 3 Watts

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS $T_A = 0°C$ to $70°C$, $V_{CC} = 5V \pm 10\%$

Symbol	Parameter	Min.	Max.	Units	Test Conditions
V_{IL}	Input Low Voltage	−0.5	+0.8	Volts	
V_{IH}	Input High Voltage	2.0	V_{CC} + 0.5	Volts	
V_{OL}	Output Low Voltage		0.45	Volts	I_{OL} = 2 mA [1]
V_{OH}	Output High Voltage	2.4		Volts	I_{OH} = −400 μA
I_{CC}	Power Supply Current		400	mA	@ $T_A = 0°C$
I_{LI}	Input Leakage Current		10	μA	$V_{IN} = 0 - V_{CC}$
I_{LO}	Output Leakage Current		±10	μA	$V_{OUT} = 0.45 - V_{CC}$
I_{LCL}	LC0, LC1, LC2 Input Low Current	−125	−450	μA	V_{IN} = 0 Volts, Reset = "1" (2)
V_{BLI}	Bus Clock Input Low Voltage	−0.5	0.8	Volts	
V_{BHI}	Bus Clock Input High Voltage	2.0	V_{CC} + 1.0	Volts	
V_{CLI}	Character Clock Input Low Voltage	−0.5	0.8	Volts	
V_{CHI}	Character Clock Input High Voltage	2.2	V_{CC} + 0.5	Volts	
V_{RLI}	Reference Clock Input Low Voltage	−0.5	0.8	Volts	
V_{RHI}	Reference Clock Input High Voltage	2.2	V_{CC} + 0.5	Volts	

NOTE:
1. I_{OL} = 2.6 mA on the $\overline{S1}$ and $\overline{S0}$ pins.
2. Measured after at least 5 BCLK cycles after RESET = High

A.C. CHARACTERISTICS

$T_A = 0°C$ to $70°C$, $V_{CC} = 5V \pm 10\%$. All timings in nanoseconds. CL = 50 pF.

82730 Bus Interface Input Timing Requirements

Symbol	Parameter	Min.	Max.	Units	Test Conditions
TCLCL	BCLK Cycle Period	125	2500	ns	
TCLCH	BCLK Low Time	52		ns	
TCHCL	BCLK High Time	52		ns	
TCH1CH2	BCLK Rise Time		30	ns	0.45V → 2.4V (1)
TCL1CL2	BCLK Fall Time		30	ns	2.4V → 0.45V (1)
TDVCL	Data in Set-Up Time	20		ns	

ORDER NUMBER: 210931-004

A.C. CHARACTERISTICS (Continued)

82730 Bus Interface Input Timing Requirements (Continued)

Symbol	Parameter	Min.	Max.	Units	Test Conditions
TCLDX	Data in Hold Time	5		ns	
TARYHCH	Async. READY Active Set-Up Time	35		ns	
TSRYHCL	Sync. READY Active Set-Up Time	20		ns	
TRYLCL	READY Inactive Set-Up Time	10		ns	
TCLRYX	READY Hold Time	20		ns	
TCTVCL	HLDA, RESET Set-Up Time	35		ns	
TCLCTX	HLDA, RESET Hold Time	10		ns	
TCAVCAX	CA Pulse Width	100		ns	
TRIVRIX	IRST Width	100		ns	
TRLLCH	LC_X Input Hold Time	5TCLCL		ns	(2)

82730 Bus Interface Output Timing Response

Symbol	Parameter	Min.	Max.	Units	Test Conditions
TCLAV	Address Valid Delay	0	55	ns	
TCLAX	Address Hold Time	0		ns	
TAVAL	Address Valid to ALE/UALE Inactive	TCLCH – 30		ns	
TLLAX	Address Hold to ALE Inactive	TCHCL – 10		ns	
TCLAZ	Address Float Delay	TCLAX	45	ns	
TAZRL	Address Float to $\overline{RD}$ Active	0		ns	
TLHLL	ALE/UALE Width	TCLCH – 10		ns	
TCLLH	ALE/UALE Active Delay	0	45	ns	
TCHLL	ALE/UALE Inactive Delay	0	45	ns	
TCVCTV	Control Active Delay ($\overline{DEN}$,$\overline{WR}$,$\overline{AEN}$)	0	70	ns	
TCVCTXW	Control Inactive Delay ($\overline{WR}$,$\overline{AEN}$)	0	80	ns	
TCVCTXD	Control Inactive Delay ($\overline{DEN}$)	5	80	ns	
TCLDOV	Data Out Valid Delay	0	50	ns	
TCLDOX	Data Out Hold Time	0		ns	
TWHDX	Data Out Hold Time After $\overline{WR}$	TCLCL – 60		ns	
TCLHV	Hold Output Delay	0	85	ns	
TRLRH	$\overline{RD}$ Width	2TCLCL – 50		ns	
TCLRL	$\overline{RD}$ Active Delay	0	95	ns	
TCLRH	$\overline{RD}$ Inactive Delay	5	70	ns	
TRHAV	$\overline{RD}$ Inactive to Next Address Active	TCLCL – 40		ns	

NOTE:
2. Applies only to test mode invocation.

A.C. CHARACTERISTICS (Continued)

82730 Bus Interface Output Timing Response (Continued)

Symbol	Parameter	Min.	Max.	Units	Test Conditions
TCLSIN	SINT Valid Delay	0	70	ns	
TRIHSIL	RINT Active to SINT Inactive		250	ns	
TCHSV	Status Active Delay	0	75	ns	
TCLSH	Status Inactive Delay	0	70	ns	
TWLWH	$\overline{WR}$ Width	2TCLCL – 40		ns	
TFLHL	Bus Float to HOLD Inactive	0		ns	

82730 Display Generator Input Timing Requirements

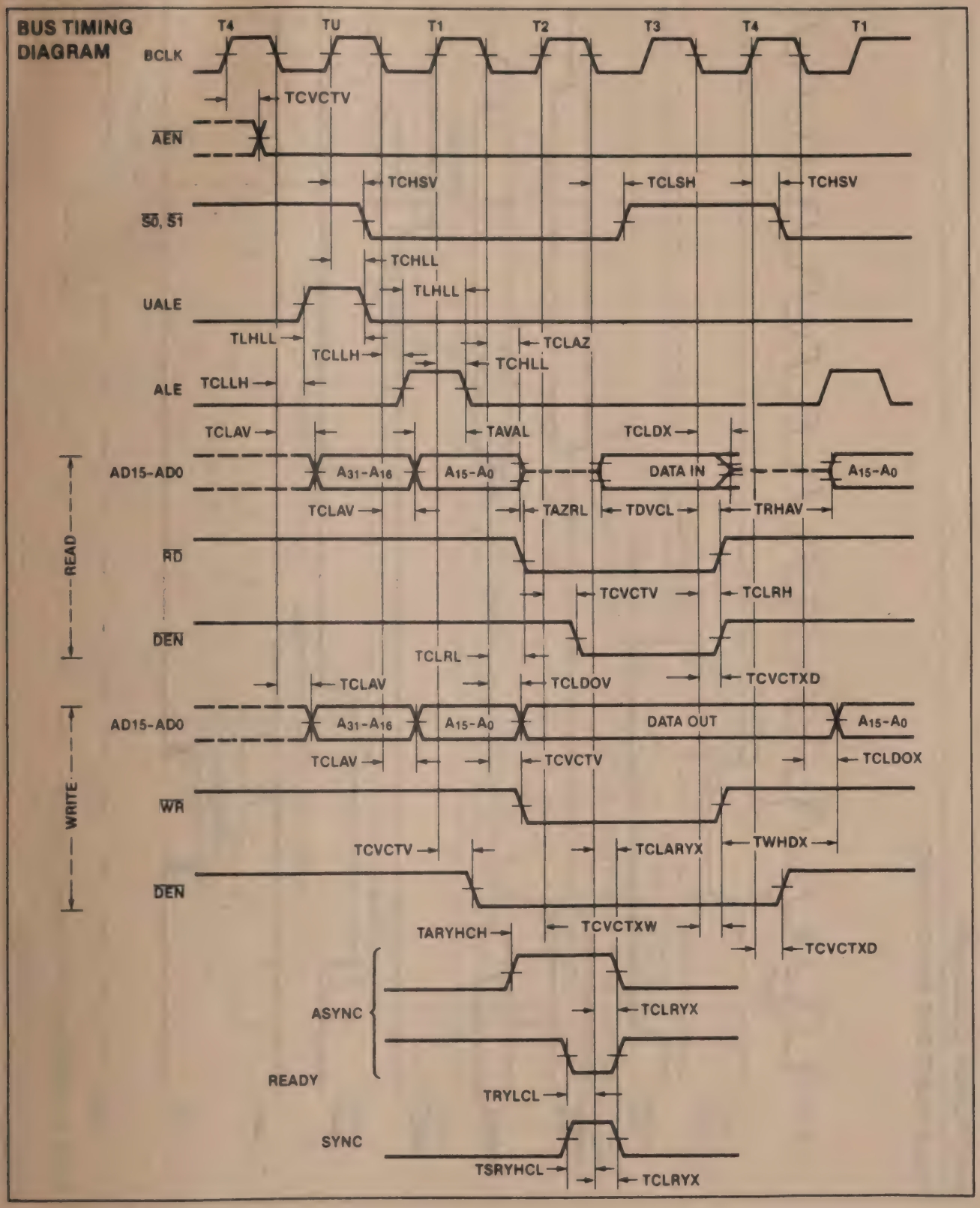
Symbol	Parameter	Min.	Max.	Units	Test Conditions
TRCHRCH	RCLK Cycle Period	100	2500	ns	
TRCHRCL	RCLK High Time	40		ns	
TRCLRCH	RCLK Low Time	40		ns	
TRRCK	RCLK Rise Time		30	ns	0.45V→2.4V (1)
TFRCK	RCLK Fall Time		30	ns	2.4V→0.45V (1)
TCCHCCH	CCLK Cycle Period	100	None	ns	
TCCHCCL	CCLK High Time	30		ns	
TCCLCCH	CCLK Low Time	40		ns	
TRCCK	CCLK Rise Time		30	ns	0.45V→2.4V (1)
TFCCK	CCLK Fall Time		30	ns	2.4V→0.45V (1)
TVCVCR	HSYNC, SYNCIN Set-Up Time	30		ns	
TCRVCX	HSYNC, SYNCIN Hold Time	10		ns	
TLPVCF	LPEN Set-Up Time	30		ns	
TCFLPX	LPEN Hold Time	10		ns	
TRCHCCH	CCLK/RCLK Skew During CSYNC	–10	10	ns	

82730 Display Generator Output Timing Response

Symbol	Parameter	Min.	Max.	Units	Test Conditions
TCCHDV	Data, Line Count and Attribute and Output Valid Delay from the Delay from the Rising Edge of CCLK		60	ns	$C_L = 100$ pF
TCCHDX	Data, Line Count and Attribute and Output Hold Time	5		ns	$C_L = 100$ pF
TRCHCV	Delay of Outputs CSYNC, VSYNC, HSYNC or RRVV from the Rising Edge of RCLK		70	ns	$C_L = 100$ pF
TCCHCL	CCLK Rising to $\overline{CHOLD}$ Low		75	ns	$C_L = 50$ pF
TRCLCH	RCLK Falling to $\overline{CHOLD}$ High		60	ns	$C_L = 50$ pF

NOTE:
1. Clock maximum rise and fall times are for functionality only. AC timings are not tested at this condition.
2. Applies only to test mode invocation.

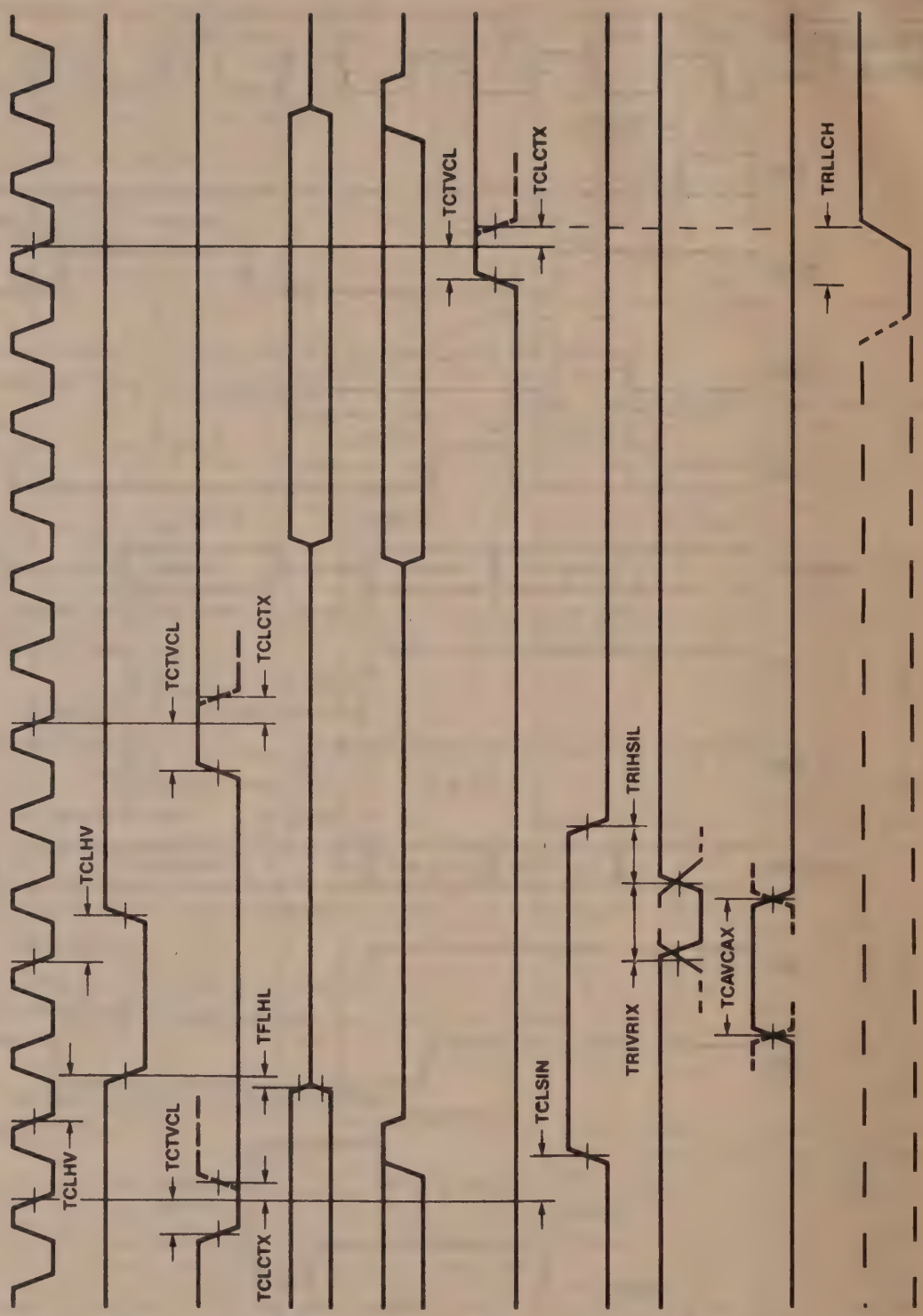
WAVEFORMS



ORDER NUMBER: 210931-004

WAVEFORMS (Continued)

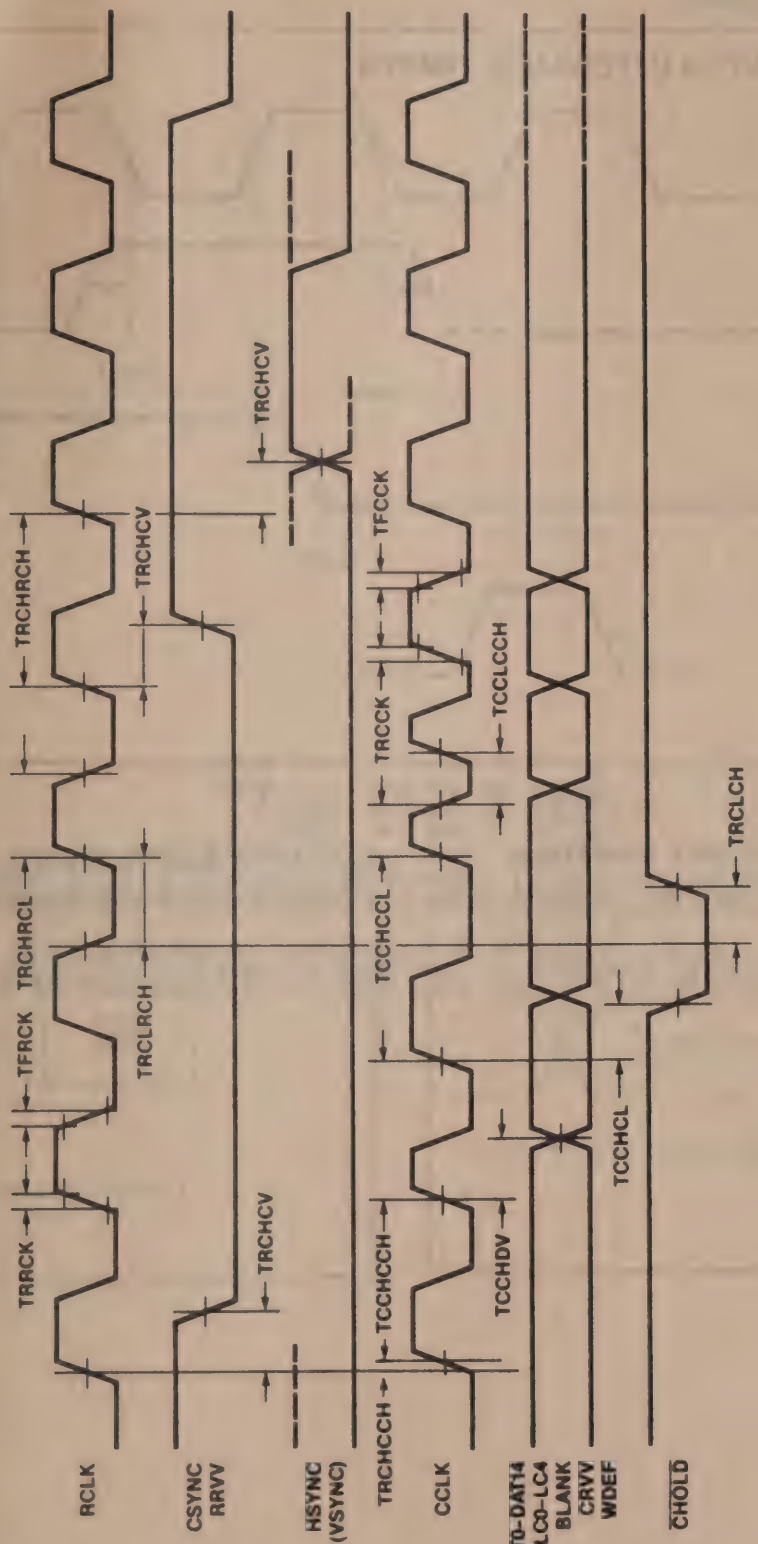
HOLD, RESET, SINT AND CA TIMING



ORDER NUMBER: 210931-004

WAVEFORMS (Continued)

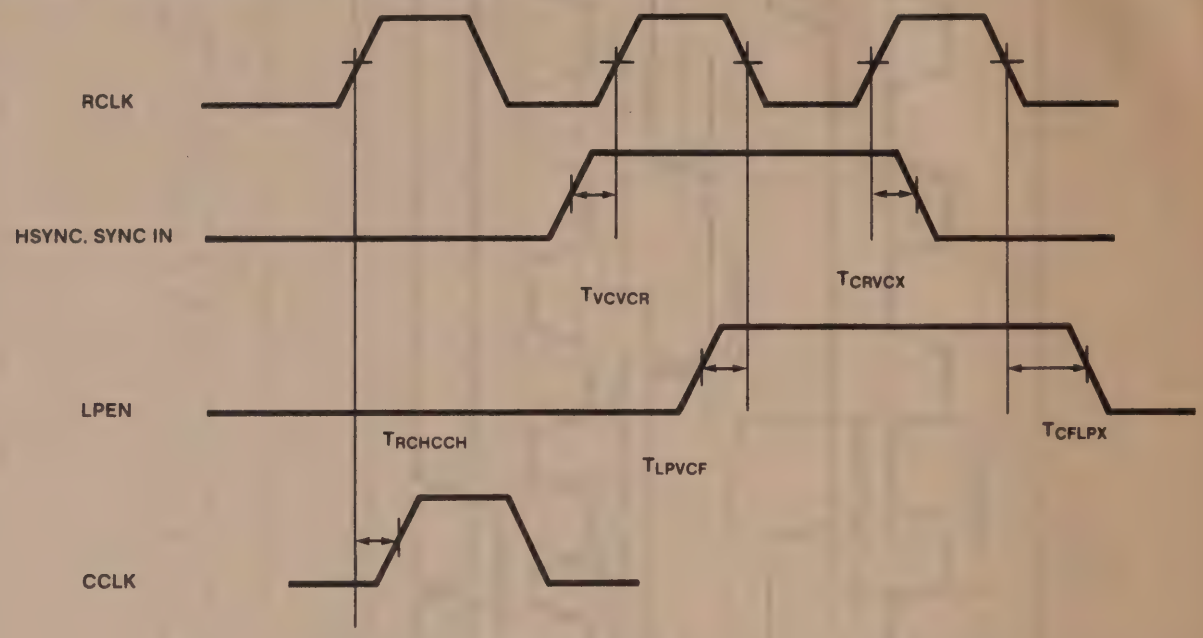
DISPLAY GENERATOR INTERFACE TIMING



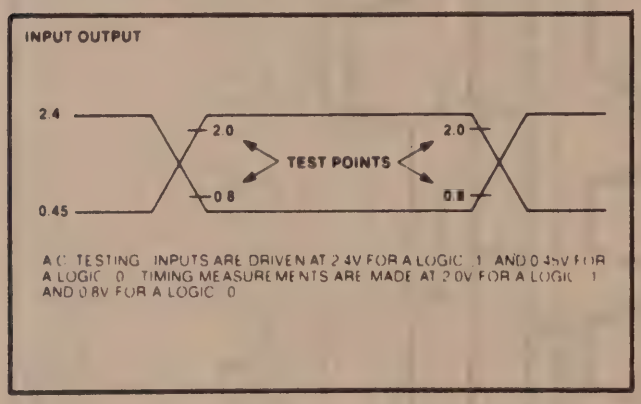
ORDER NUMBER: 210931-004

WAVEFORMS (Continued)

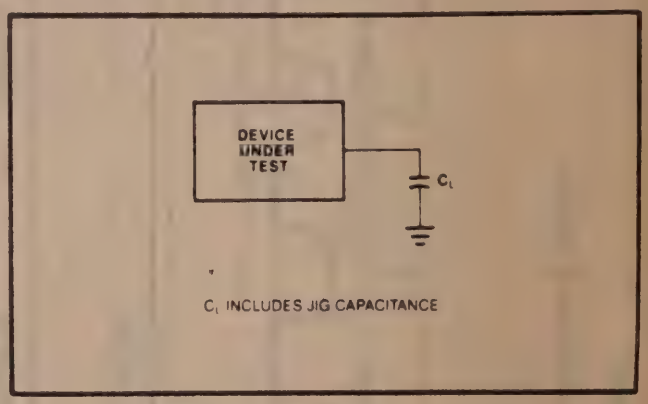
DISPLAY GENERATOR INTERFACE TIMING



A.C. TESTING INPUT, OUTPUT WAVEFORM



A.C. TESTING LOAD CIRCUIT



intel

APPLICATION NOTE

AP-259

November 1985

The 82786 CHMOS Graphics Coprocessor Architectural Overview

Order Number: 122711-001

PREFACE

82786 FEATURES AND PERFORMANCE

The 82786 is a powerful, yet flexible component which will be a candidate as a standard for microcomputer graphics applications including personal computers, engineering workstations, terminals, and laser printers. Its advanced software interface contrasts sharply with existing products by making applications and systems level programming efficient and straight-forward. Its performance and high-integration make it a cost-effective component while improving the performance of nearly any design.

The following list is a summary of the 82786's capabilities (assuming 10 MHz system clock and 25 MHz video clock):

Windows:	Practically unlimited support
Colors:	Up to 1024 displayable simutaneously with support for 4 external color palettes
Lines, Polylines, Polygons:	2.5 Million pixels per second
Circles, Arcs:	2.0 Million pixels per second
Fills:	Supported via horizontal line command (30 Million bits per second)
Bit Block Transfer:	24 Million bits per second
Bit-map Memory:	Up to 4 MBytes of directly accessed DRAM
Resolution:	Up to 200 MHz monitors supported; this is equivalent to configurations such as 640 x 480 x 8 or 1024 x 1024 x 2 @ 60 Hz (non-interlaced); up to 4096 x 4096 x 1 or 2048 x 1536 x 8 with video DRAMs.
Zoom:	1 to 64 times vertical and horizontal
Character Drawing:	25 thousand per second with colors, path, and rotation attributes
Character Fonts:	Unlimited number from bit-map or system memory
Character Size:	16 x 16 maximum hardward size; unlimited with bit-block transfer
Scroll, Pan:	Instantaneous in any direction with no external logic

The performance of the 82786 is of little value without applications and system-level software to use it. Customers can write their own software following the suggestions of the 82786 Software Interface Applications Note or the appropriate third-party vendors' software packages. Intel has evaluated several major products and presently recommends Microsoft Windows™, Digital Research GEM™, Novagraphics Nova CGI and GKS™, and Graphic Software Systems CGI and GKS™, Window Manager™, and GKS™. These packages appear to be easily adapted to 82786-based systems, are likely to emerge as de facto industry standards, and would permit a wide array of applications to run with little or no modification on 82786-based products.

For more information on these products, please contact these vendors directly:

Digital Research, Inc.
P. O. Box DRI
Monterey, CA 93942
(408) 649-3896

Graphic Software Systems
P. O. Box 673
Wilsonville, OR 97070
(503) 682-1606

Microsoft Corporation
Box 97200
Bellevue, WA 98009
(206) 828-8080

Novagraphics International Corporation
1015 Bee Cave Woods
Austin, TX 78746
(512) 327-9300

The 82786 was designed to permit compatibility with de facto hardware standards. Use of the 82786 with appropriate Intel microprocessors permits the design of systems which can emulate the family of IBM™ personal computer products. The 82786's support of the IBM Color Graphics Adapter-compatible bit-map eases the task of running existing applications software on new video hardware.

For details please refer to the 82786 PC Compatibility Applications Note. Additional documentation available for the 82786 includes the Data Sheet, the User Manual and Application Notes.

For all questions, clarifications, or requests for additional documentation please contact your local Intel sales office or authorized distributor.

CHAPTER 1
INTRODUCTION

1.1 OVERVIEW

This document provides the reader with an introduction to the architecture and key features of the Intel 82786 Graphics Coprocessor from Intel. The 82786 serves such applications as graphics terminals and work stations, personal computers, printers, and other products requiring the capability to create, store, and output bit-map graphics.

The 82786 works with all Intel microprocessors, and is a high-performance replacement for sub-systems and boards which have traditionally used discrete components and/or software for graphics functions. The 82786 requires minimal support circuitry for most system configurations, and thus reduces the cost and board space requirements of many applications. The 82786 is based on Intel's advanced CHMOS process.

The advanced performance and ease-of-use of the 82786 make it a candidate for an industry standard for applications in microcomputer graphics markets. Some of the leading features of the 82786 are:

- Fast polygon and line drawing
- Hardware windows
- High speed character drawing
- Interface designed for device independent software standards
  - — Virtual Device Interface
  - — Graphics Kernal System
  - — NAPLPS
- Advanced DRAM controller for graphics memory up to 4 Mbytes
- Fast bit-block copies between system and bit-map memories
- Supports up to 200 MHz CRTs or higher
- Up to 1024 simultaneous colors per frame
- Programmable video timing
- High Integration
- Third-party software support
- 88 pin leaded chip carrier and pin grid array
- Provides support for rapid filling with patterns
- IBM Personal Computer Color Graphics Adapter-compatible bit-map
- International character support
- Advanced CHMOS technology
- Integral video DRAM support

1.2 ARCHITECTURAL MODEL

The 82786 architecture fits with traditional computer graphics models. A typical subdivision of the tasks is:

- Graphics task partitioned into:

  Drawing (line, polygons, characters, block image copies)

  Windowing (concurrent windows on the screen)

  Refresh (CRT timing, video data output)
- Typical integrated solutions to these functions have been:

  First generation IC: 6845, 8275 - refresh

  Second generation LSI: 82720 - drawing + refresh

  Third generation VLSI: 82786 - drawing + windowing + refresh

The 82786 is a co-processor with two separate on-chip processing units, the graphics processor and display processor, which operate concurrently with the system CPU. Instructions to the display and graphics processors are placed in memory by the CPU. Registers on the 82786 are dedicated to pointing to the starting addresses of the first memory blocks of instructions controlling the on-chip processors, and each memory block points to subsequent blocks in a linked-list architecture. Access by the CPU to these registers may be I/O- or memory-mapped, and portions of memory may be shared between the 82786 and the CPU.

1.3 BIT MAPS AND WINDOWS

The 82786 concepts of "bit maps" and "windows" are based upon definitions from the ANSI work on windows.

The 82786 can create and maintain multiple sets of graphics images in memory. These sets of images in memory are called "bit maps". 82786 can combine subsets of these bit-maps into a viewable, multi-region display screen. Each of these separate areas on the screen are called "windows".

Most graphics systems today use software to generate a bit-map representation of the full contents of the display called a "frame buffer". The 82786 uses a high-level window descriptor list and specialized hardware to generate the screen contents using portions from separate bit maps of memory (Figure 1-1). This permits the display to be instantaneously altered, eliminating the time required to update a similar frame buffer image using software alone.

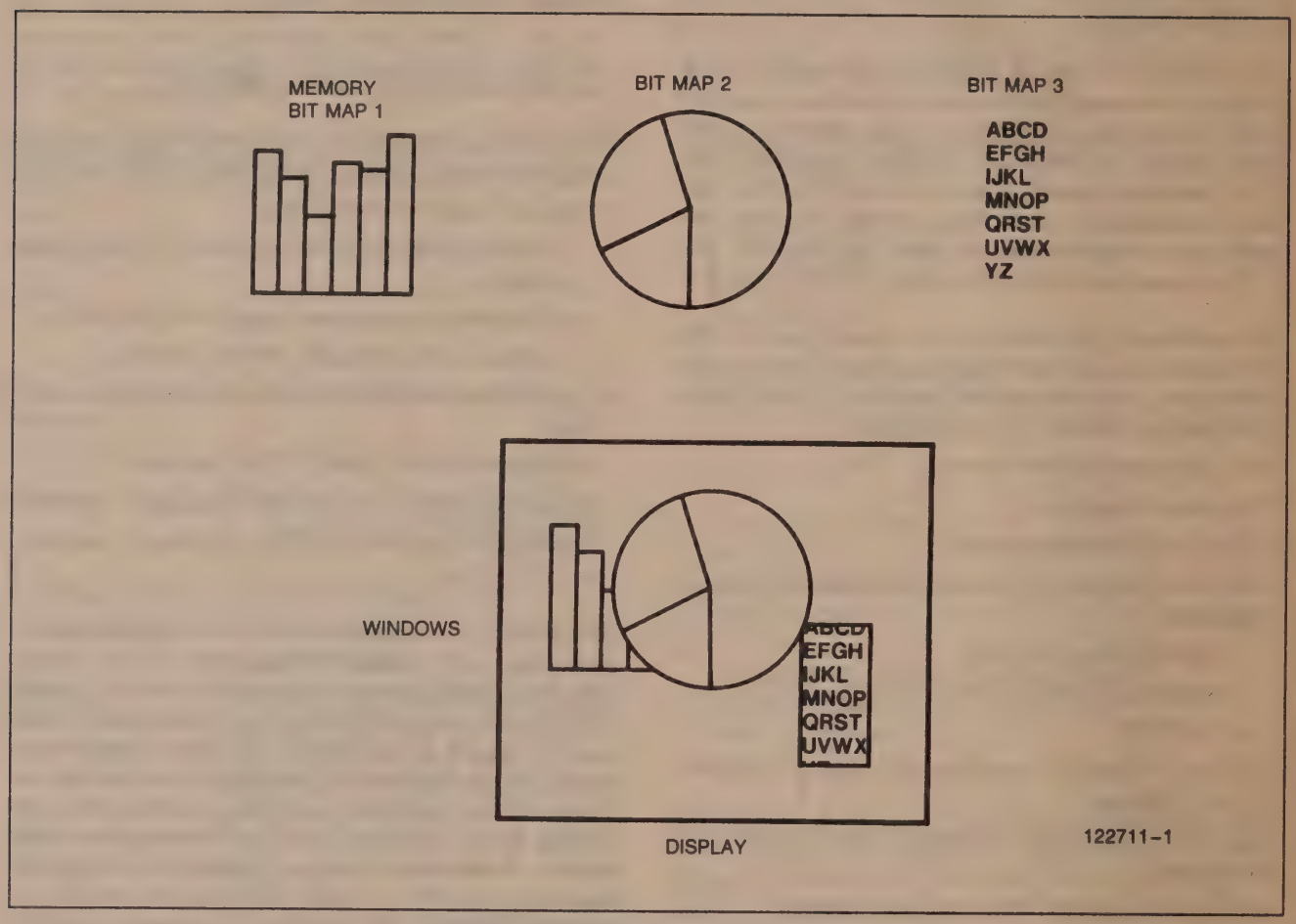


Figure 1-1. Bit Maps and Windows

1.4 FUNCTIONAL OVERVIEW

The 82786 performs many functions within a single integrated circuit. Figure 1-2 identifies a block diagram of the component and explanations of each function module.

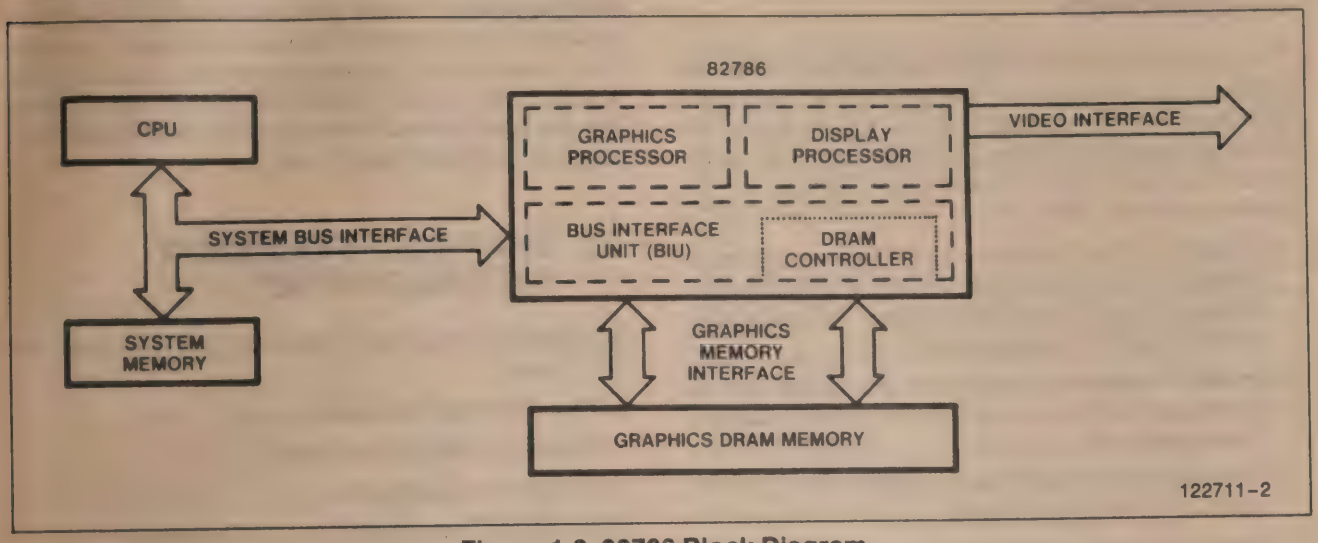


Figure 1-2. 82786 Block Diagram

The major functions of each block are:

- Graphics Processor (GP): — draws lines, circles, polygons, and other primitives
  — draws characters
  — executes block image manipulation instructions
- Display Processor (DP): — manages windows including zoom
  — provides cursor
  — refreshes screen (up to 200 MHz dot rate)
  — loads shift register of video DRAMs
- DRAM Controller: — controls up to 4 Mbytes of interleaved graphics memory including page-, static column-, and fast page-mode DRAMs (interleaved or non-interleaved banks)
- BIU: — allows the CPU to access the graphics memory and the 82786 to access the system memory

CHAPTER 2
GRAPHICS PROCESSOR

2.1 OVERVIEW

The graphics processor creates and updates all of the graphics and text in each of the bit maps within graphics memory. It is responsible for all of the geometric drawing, character drawing and image movement within and between the bit maps. Some features of the graphics processor are:

- permits bit maps to begin at any word in system or graphics memory; only one bit map is active for GP drawing at one time although many bit maps may reside in memory simultaneously.
- permits bit maps to be any size (up to 32K x 32K pixels) and use 2, 4, 16, or 256 colors (i.e. 1, 2, 4, or 8 bits per pixel);
- draws geometric shapes with attributes such as texture and color, into bit maps;
- draws characters with attributes such as color, path, rotation, and proportional spacing using user-defined fonts into bit maps;
- combines one rectangular portion of an image with another area, within the same bit map or into another bit map. (BIT Block Transfer or Bit-Blt);
- all drawing allows logical operations between source and destination (for example Exclusive-Or of the Complement of Source with Destination);
- all drawing can be clipped to a rectangular region;
- supports picking, a mechanism for advanced user interfaces which allows the issuing commands via the selection of "graphic menus" (called icons) by manipulating pointing devices.

The Graphics Processor fetches its instructions directly from a linked list in memory which is created and updated by the CPU. The initial address of the list is contained in a dedicated register in the 82786 and the addresses of subsequent instructions are pointed to by the contents of previous instructions. Each instruction contains a bit which indicates to the graphics processor that it should stop (if set) and await new instructions. More detail on the command format is given in section 2.8 "Graphics Processor Command List Format."

2.2 BIT MAPS

All graphics and text creation is written into bit maps. Bit maps are rectangular drawing area composed of bits of pixel-oriented memory. The bit maps may be up to 32,000 pixels in each direction and contain from one to eight bits of color or gray scale information. Bit maps may be started on any even address in the 4 Mbyte space and the number of bit maps in memory is unlimited (except by the amount of memory available). The variable bits per pixel feature permits the use of several bits per pixel for multicolor graphics while using only a single bit per pixel for efficient text memory.

2.3 GRAPHICS PROCESSOR INSTRUCTION SET

The graphics processor instruction set is divided into five classes:

1. Non-Drawing Commands
2. Drawing Control Commands
3. Geometric Commands
4. Bit Block Transfer (BIT-BLT) Commands
5. Character Block Transfer (CHA-BLT) Commands

2.3.1 Non-Drawing Commands

The first class of commands are used to control the method in which the commands are fetched. Also included in this list are commands to load and dump 82786 internal registers. These commands are:

- NOP - No Operation
- LINK - Link To Next Command (Unconditional Jump)
- ENTER_MACRO - Enter Macro (Subroutine Call)
- EXIT_MACRO - Exit Macro (Subroutine Return)
- INTR_GEN - Generate Interrupt
- DUMP_REG - Dump Internal Register
- LOAD_REG - Load Internal Register

2.3.2 Drawing Control Commands

The graphics processor works in only one bit map and with one set of attributes at a time. The graphics processor maintains an imaginary cursor, GCPP (Graphics Current Position Pointer), which points to a particular position (x, y coordinates) within the bit map from which all relative coordinates are calculated. The GCPP is updated at the end of each drawing command.

The following commands are used to define the current bit map and attributes and set the Current Position Pointer:

- DEF_BIT_MAP - Define Bit Map
- DEF_CLIP_RECT - Define Clip Rectangle (see 2.4)
- DEF_COLORS - Define Colors
- DEF_TEXTURE - Define Texture
- DEF_LOGICAL_OP - Define Logical Operation (see 2.6)
- DEF_CHAR_SET - Define Character Set
- DEF_CHAR_ORIENT - Define Character Orientation
- DEF_CHAR_SPACE - Define Inter Character Spacing
- ABS_MOV - Absolute Move GCPP
- REL_MOV - Relative Move GCPP
- ENTER_PICK - Enter Pick Mode
- EXIT_PICK - Exit Pick Mode

2.3.3 Geometric Commands

These commands allow the 82786 to draw points, lines, and arcs in a variety of ways:

- POINT - Draw Point
- INCR_POINT - Draw Incremental Points
- CIRCLE - Draw Circle
- LINE - Draw Line
- RECT - Draw Rectangle
- POLYLINE - Draw Polyline
- POLYGON - Draw Polygon
- ARC - Draw Arc
- SCAN_LINES - Draw Series of Horizontal Lines

2.3.4 Bit Block Transfer (Bit-Blt) Commands

These commands allow rectangular image pieces to be combined from piece of bit-map memory to another. The graphics processor automatically inserts the new data in the correct order in the destination so that each line of pixels remains consecutive for both existing and new data.

- BIT_BLT - Bit Block Transfer within bit map
- BIT_BLT_M - Bit Block Transfer between bit maps

The command specifies the origin of the source rectangle as well as the height and width. The destination origin is the GCPP coordinates. For Bit-Blts between bit maps, the destination is the active bit map and the memory address of the source origin and source bit map size is specified. Bit-Blts between bit maps can only use bit maps with the same number of bits per pixel.

2.3.5 Character Command

This command allows character fonts stored in memory in pixel form to be drawn into the bit map by an application using character codes such as ASCII:

- CHAR - Draw Character String

The CHAR command defines transparency/opaqueness for a character string, the pointer for the character string, and the number of character in the string. The pixel contents of the character to be drawn may be located anywhere in the memory space of the 82786 and accessed with either an 8- or 16-bit reference to the specific character. The string range specifies the 8- or 16-bit references for each character to be drawn. Section 2.7 discusses the use of character fonts.

Standard character fonts can be flexibly drawn because path and rotation are defined with a DEF_CHAR_ORIENT command and inter-character spacing is defined with a DEF_CHAR_SPACE command. This permits the variable spacing of text, direction of text, and rotation of characters to be specified by the application without making alteration of the font necessary. Simple one-bit per pixel character font definitions can be used in color applications because foreground and background colors are specified by the DEF_COLOR command and the necessary bits are written for each pixel during the drawing process.

2.4 DRAWING ATTRIBUTES

A drawing operation refers to the act of modifying pixels within a bit map during the execution of the GP commands. All drawing that the GP performs (including lines, arcs, characters and Bit-Blts) is subject (with exceptions noted) to six attributes which should be defined before any drawing commands are executed. The attributes are:

1. Pixel Plane Mask;
2. Logical Operation;
3. Clipping Rectangle;
4. Foreground and Background color (not applicable to Bit-Blt);
5. Transparent or Opaque mode (not applicable to Bit-Blt);
6. Pattern mask of 16 bits (not applicable to Bit-Blt or characters).

The pixel plane mask is helpful in restricting the graphics primitives to update a subset of the bits per pixel.

This permits one set of drawings to exist in one or more colors and allow other text or graphics information to reside in different color bits of the same bit map. Raster operations can be used to combine existing pixel information in the bit map with the new pixel information generated as a result of the new drawing operation, such as displaying only the overlapping regions of two shapes. The clipping rectangle limits the effects of drawing operations to a subset of the bit map.

Foreground and background colors set the two colors drawn by all drawing operations (if both are needed). The transparent mode draws only the foreground color into the bit map (for dotted lines or characters) and leaves the pixels between the dots or characters unchanged. The opaque mode draws the foreground color and fills in the background color between the dots or characters. The pattern defined in the mask cause a logical operation with drawing commands and permit dotted and dashed lines, arcs, and other shapes. DEF__PATTERN sets transparent/ opaque for drawing operations other than character, which is defined in CHAR.

2.5 CLIPPING

The clipping rectangle is used to prevent drawing outside a specified rectangular region. The clipping rectangle can be any rectangle within a bit map or the entire bit map. Pixels are not drawn beyond the limits of the clipping rectangle and characters which would be partially clipped are not drawn at all.

In a special mode, "pick mode," the clipping rectangle is used to perform a different function. The clipping rectangle may be controlled by software to support the selection of objects on the display with a pointing device. When in pick mode the drawing commands are executed but pixels are not updated in memory. Instead, a flag is set in a register if any of the pixels generated by the command lie within the clipping rectangle. In this way it is easy to set the clipping rectangle to correspond to the location of a graphics pointing device (such as a mouse) and re-process the graphics command list to find which drawing command corresponds to the selected area.

2.6 LOGICAL OPERATION

The logical operation is an attribute that applies to all subsequent pixel update operations (line, arc, character, Bit-Blt etc.). It is an operation which can logically combine the contents of separate bit-map locations to produce new bit-map patterns. All sixteen binary functions are permitted between both the source and destination.

- AND
- OR
- EXCLUSIVE-OR

Six of the combinations provided are special:

- REPLACE destination with source
- REPLACE destination with complement of source
- SET all destination bits to 0
- SET all destination bits to 1
- REPLACE destination with complement of destination
- REPLACE destination with destination (NOP)

2.7 CHARACTER FONTS

The Graphics Processor supports an unlimited number of character fonts, that can reside anywhere in the 4 Megabyte address space. The character string to be written can be defined either as a string of bytes or as a string of words depending upon the type of font used. The active font type and upper and lower memory addresses of the font to be used are set via the DEF__CHAR__SET command.

Each character in the character font has an independently programmable size of up to 16 by 16 pixels, allowing individual characters to have different sizes for proportional spacing. Each character resides in a block containing n + 1 words of memory where n is the pixel height of the character. The first word contains fourteen bits to define the height and width of the character. The remaining two bits specify if the following character should be an overstrike or if the character exceeds sixteen pixels in either dimension to cause a software trap. Overstriking is useful for efficient implementation of underline and accents, and prevents updating the GCCP after the character is drawn.

For larger characters than 16 by 16, the trap bit in the font can cause an interrupt to the CPU so that software can specially process that character such as a Bit-Blt. The perception of larger characters than 16 by 16 can also be created by dividing characters into subsets such as quadrants, and executing multiple character drawing commands. Software use of the DEF__CHAR__SPACE command supports negative inter-character spacing to permit kerning, such as for italic fonts.

The byte or word strings used as parameters for the CHAR command are used in conjunction with the 22-bit pointer defined in a register by the DEF__CHAR__SET command. Use of 16-bit, or word-mode, characters causes an add between the 22-bit pointer and the 16-bit reference value to access the starting address of the specific character. Because maximum character block size is seventeen words of data, approximately four thousand characters may be contained in one 16-bit font (worst case). Supplementary software in the form of a look-up table can be used to access as many as 65,000 characters in a single font. Bit-Blt can move characters of unlimited size.

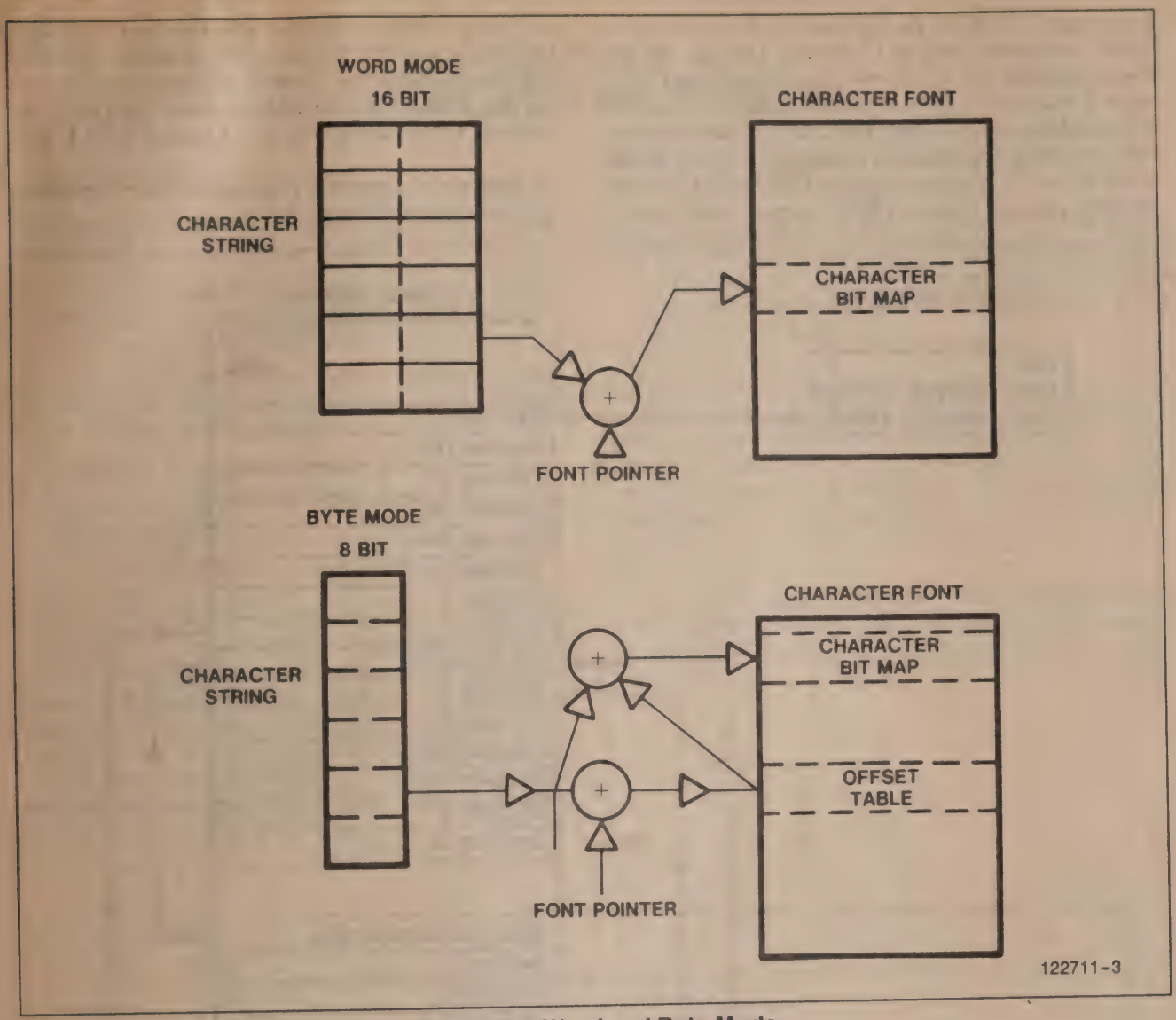


Figure 2-1. Word and Byte Mode

Use of byte-mode permits eight bit references to characters. This is important to permit existing software using ASCII and EBCDIC to be converted to 82786-based systems. 256 words of the font are reserved for a look-up table. Adding the 8-bit string parameter to the font pointer determines the word for the specific character within this table. The word is then added to the pointer to locate the character information in the font. Byte-mode permits only 256 characters in each 8-bit font. Figure 2-1 shows a description of word and byte mode.

2.8 GRAPHICS PROCESSOR COMMAND LIST FORMAT

The graphics processor executes a sequence of instructions resident in memory and runs only when an application needs to change bit-map contents or support some special function such as picking. The general format of an instruction is shown in Figure 2-2.

15 14 13 12 11 10 09 08 07 06 05 04 03 02 01	00
OPCODE	GECLbit
PARAMETER 1	
PARAMETER 2	
PARAMETER N	

Figure 2-2. Instruction Format

Each opcode resides in the high byte of the word with a GECL (Graphics End of Command List) bit in the least significant bit of the low byte and followed by a varying number of parameters in consecutive words. The graphics processor tests the GECL of each instruction and sends the graphics processor into Poll Mode when set to "1" for any opcode. Poll mode halts the graphics processor until a LINK command and upper- and lower-memory values for a link address are loaded into three reserved registers. The graphics processor then begins executing a new linked-list of instructions starting at the specified address when the GECL bit with the LINK instruction in the register is reset to 0.

An example of a graphics command block using linked-lists is shown in Figure 2-3.

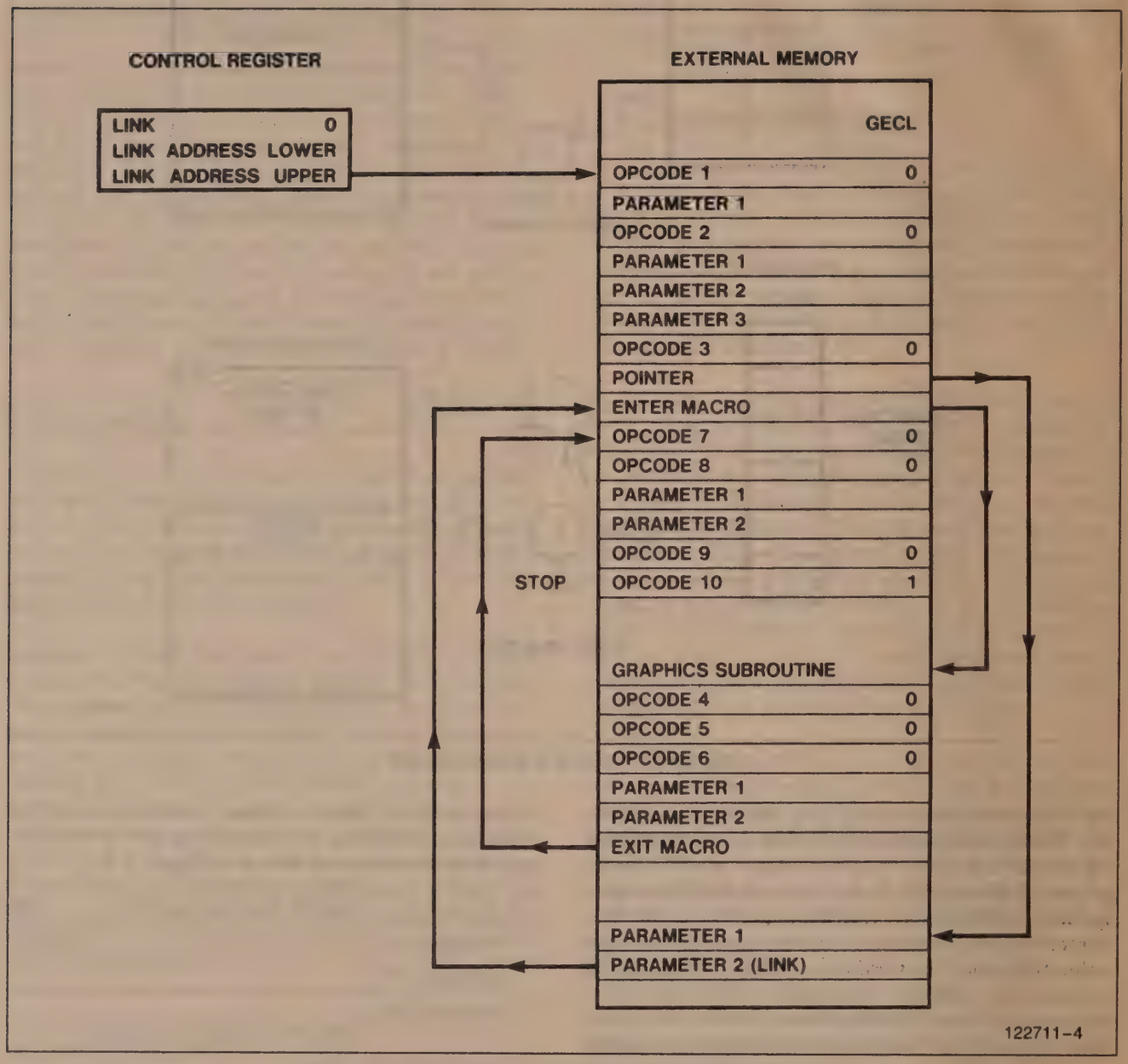


Figure 2-3. Graphics Processor Command Block

CHAPTER 3
DISPLAY PROCESSOR

3.1 OVERVIEW

The display processor has five main functions in generating the display contents for output:

1. To retrieve the memory contents of selected bit maps and output corresponding pixels into separate regions on the display screen (windows);
2. To permit selected portions of bit maps to be magnified on the display (zooming) horizontally and/or vertically via pixel replication;
3. To provide a "pointing symbol" (cursor);
4. To generate control and video data signals to the display hardware;
5. Load the shift registers of video DRAMs.

Control of the display processor is programmed via on-chip registers. Content of the display is dynamically altered by the application (or system software) without causing unacceptable display blinking. Using memory-mapped CPU alteration of parameters, the DP will load the register set with the new parameters during vertical retrace. By altering the registers to point to a new display list, the change of display lists can occur between refresh cycles.

3.2 WINDOWS

Windows are the portions of bit maps which are output by the display processor. Up to 16 window segments or tiles can be displayed on the same scan line of the CRT, while there may be as many windows vertically as the number of scan lines.

The 82786 treats the screen as divided into horizontal strips (Figure 3-1) of arbitrary width, where the horizontal format of window tiles across the strip remains constant for the whole strip. This divides the region into rectilinear areas, which are easy to manage. By combining strips, overlapping windows can easily be obtained.

Windows may essentially be arbitrarily shaped (circular, irregular, etc.) because a new strip may be defined every display line, similar to the format shown in Figure 3-2.

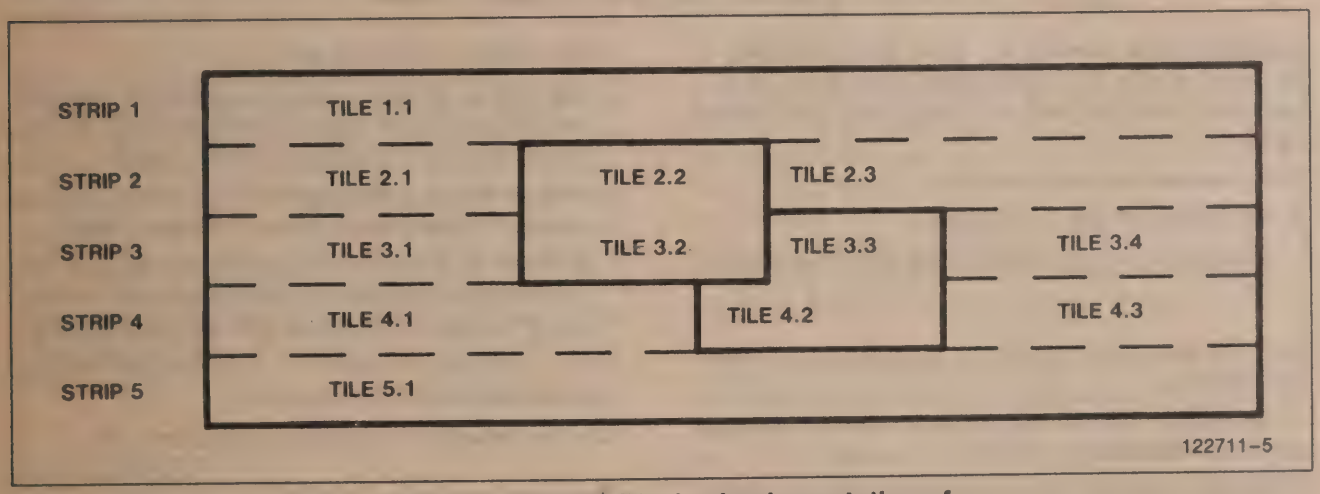


Figure 3-1. Sample Display Implementation of Two Overlapping Windows

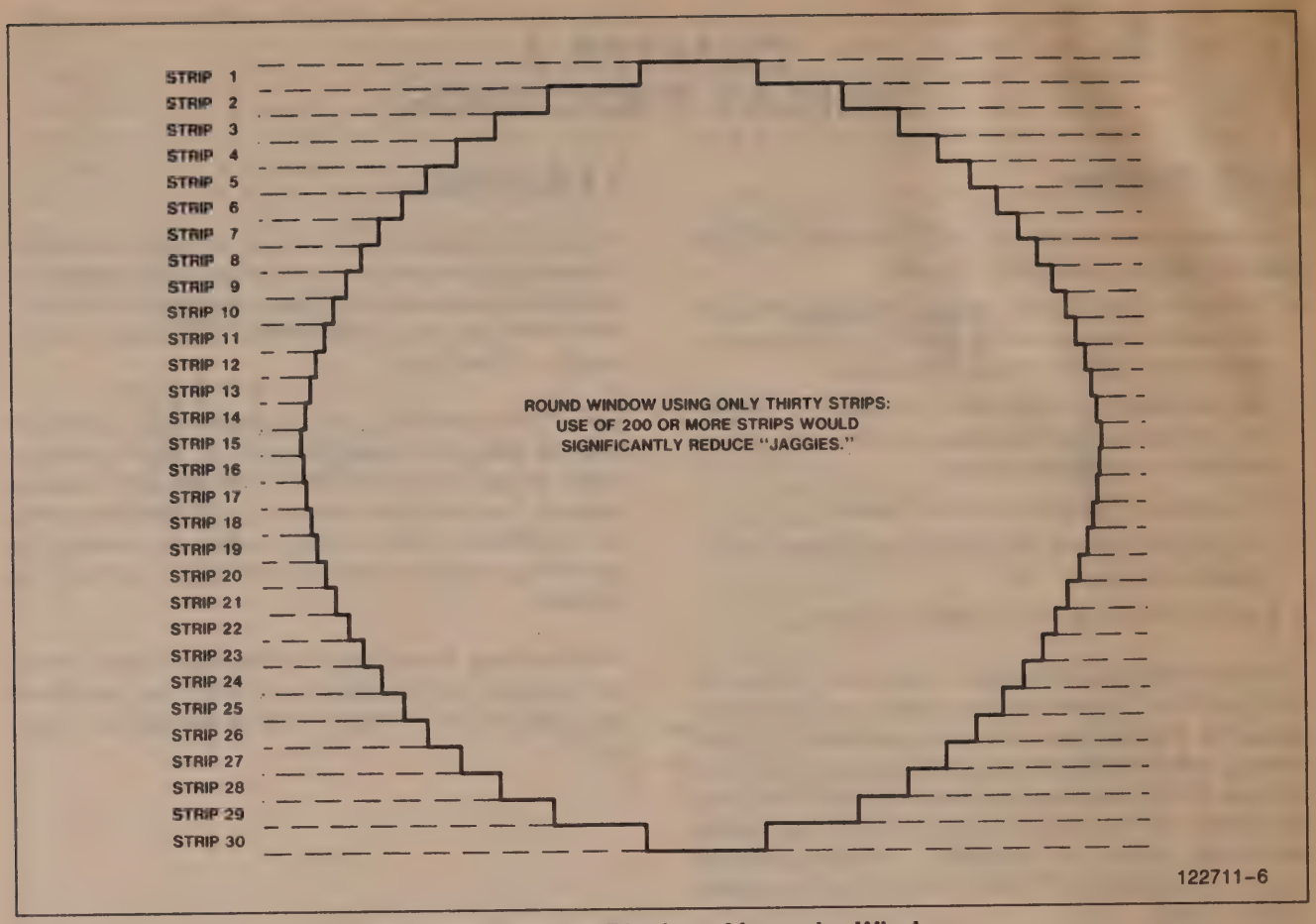


Figure 3-2. Sample Display of Irregular Window

The information needed for the display processor is contained in strip descriptor tables, each made up of a header and one or more tile descriptors. The header contains:

- the number of lines in the strip;
- the number of tiles in the strip;
- upper and lower addresses of the next strip descriptor

Each tile descriptors (which are consecutive in memory) contains:

1. the width of the bit map from which the window is being retrieved (in words);
2. the start address of the bit-map data to be displayed (word in memory and first bit location);
3. the number of words to fetch for the tile;
4. the first and last bit locations of the bit-map data to be displayed;
5. the number of bits per pixel;
6. four bits to indicate border presence for top, bottom, left, and right edges (1 indicates show border, 0 indicates show bit-map for those pixels);
7. window status information which can be used to select color palettes or other attributes (2 bits);
8. two bits to indicate bit-map configuration is byte rather than word-oriented with byte order switched and if bit-map is non-linear (for PC compatibility);
9. bit to indicate if window is to be zoomed by pixel replication of the bit-map data;
10. bit to indicate if tile if field background data.

A one-pixel border can be displayed on any or all sides of each viewport tile. This border color is defined in an 8-bit register and is the same user-definable color for all windows. Borders may be turned on or off for individual tiles.

In the absence of windows, the field background color is displayed. This single color is definable by the user in an 8-bit register. The use of background on the display minimizes system bandwidth because data is only fetched for windows and not for background, and thus saves bit-map memory.

The display processor provides padding bits when bit maps to be displayed have fewer bits/pixel than the hardware display, with no performance decrease. This allows windows of various bits/pixel to be shown simultaneously on the same display. The user programs the desired 8-bit color patterns into three registers, one serving to map each of 1-, 2-, and 4-bits per pixel information into full colors on the display.

All video output from the 82786 can be defined to begin and end at any pixel (except when in accelerated mode using external shift logic). This includes the positioning of every window and the cursor.

The display processor instruction list is controlled by the CPU. The double-word location of the first strip descriptor block is located in a register. The locations of subsequent strip descriptor tables are based upon a linked-list architecture and are provided in the preceding descriptor table. This descriptor linked-list needs only to be updated by the CPU when the window arrangement on the screen changes. New strips and segments are easily inserted into the display list by simply modifying the linked-list pointers of the preceding strips or segments.

The use of redundant lists is possible because the description of a typical display is memory-efficient and requires only about 1,000 bytes. This would permit the CPU to alter the contents of one list while the second is being used to control display processor. When the creation of the new list is complete, the registers pointing to the first strip descriptor table may be switched to the locations for the new list during vertical retrace. This permits the application to alter the display list without causing temporary swimming or blinking of the display.

3.3 CURSOR

The display processor supports a single hardware cursor which may be up to 16 x 16 pixels. This cursor may be positioned by the user anywhere on the screen. The cursor may be defined to be transparent or opaque, and may be either a block cursor or a cross-hair cursor one pixel across stretching the width and height of the screen. The color of the cursor is user-definable, as is the block cursor's pattern. Eight bits of register memory define the color and sixteen 16-bit words of register define the pattern, which is then padded with the cursor color register. Support for a blinking cursor is provided with a register for CURSOR_ON which can be toggled by the CPU as often as necessary to cause an appropriate blink rate. Multiple cursors can be simulated by drawing them in software, especially using bit-blt.

3.4 ZOOM

The display processor allows selected windows to be zoomed (using pixel replication) up to 64 times horizontally and vertically (independently, in steps of one). The setting of the zoom bit in the tile descriptor table causes replication of the pixels in memory according to horizontal and vertical scaling factors contained in registers.

3.5 VIDEO INTERFACE

Eight parallel video data output lines provide video output which may be used as eight bits pixel on the CRT, or externally shifted to boost maximum display resolution. The dot rate output is controlled by an independent video clock which may be up to 25 MHz. Horizontal signals are programmable from 1 to 4096 cycles of the video clock and vertical sync signals from 1 to 4096 scan lines. Use of eight external video data pins allow up to 256 different colors to be directly displayed. Other CRT control lines provided by the display processor are VSYNC, HSYNC, BLANK.

Several 82786s can be used together for higher performance graphics. For multiple 82786 Systems, one 82786 acts as a master generating VSYNC and HSYNC, and the other 82786s act as slaves using the master synch signals for timing through the use of their own VSYNC and HSYNC as inputs. Each 82786 has its own bit-map memory with separate graphics processor lists to form a bit-plane architecture, but use the same display list. The BLANK signal is not used by slave 82786s.

External color palettes are supported, and, by use of the two window status lines, the application may select one of four color combinations for any window. This supports a maximum of 1024 simultaneous colors per frame. The palette may be programmed by latching the default video data when the BLANK pin is high. The display processor can support non-interlaced, and interlaced-synch displays. Selection of the interlacing, control to support external shifting of the video data, default video data contents, and slave/master status for each 82786 are controlled via dedicated registers. The 82786 may be synchronized to an external source ("Gen-Locking").

CHAPTER 4
82786 SYSTEMS

4.1 TYPICAL SYSTEM CONFIGURATIONS

The 82786 can be used in many different configurations, each providing cost and performance appropriate for different applications and markets.

Three typical applications in which the 82786 could be used are:

1. Low-priced personal computer (Figure 4-1);
2. Multi-tasking office workstation (Figure 4-2);
3. High-performance workstation for processing-intensive, high-resolution applications in engineering (Figure 4-3).

4.2 DRAM CONTROL

The DRAM controller on the 82786 supports an array of up to 32 memory chips without extra logic and up to a 4 megabyte address space. DRAMs supported have densities ranging from 8K to 1 megabit and organiza-

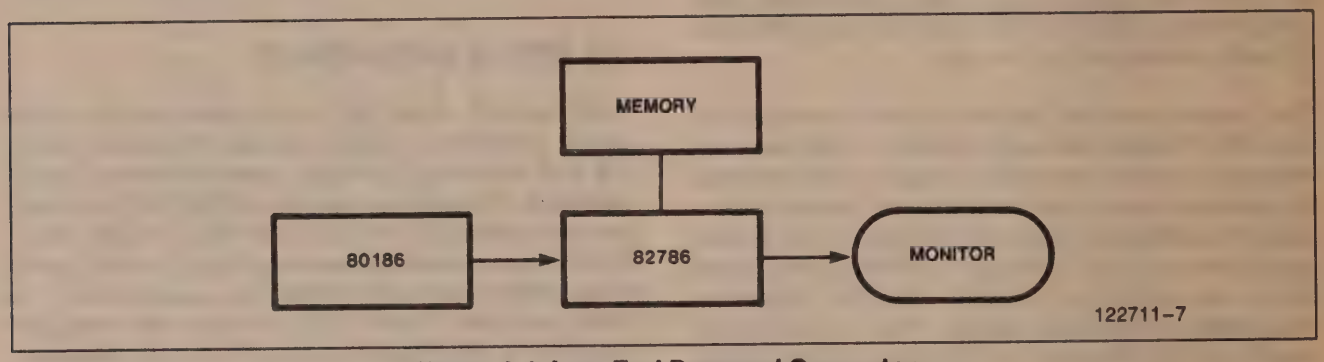


Figure 4-1. Low End Personal Computer

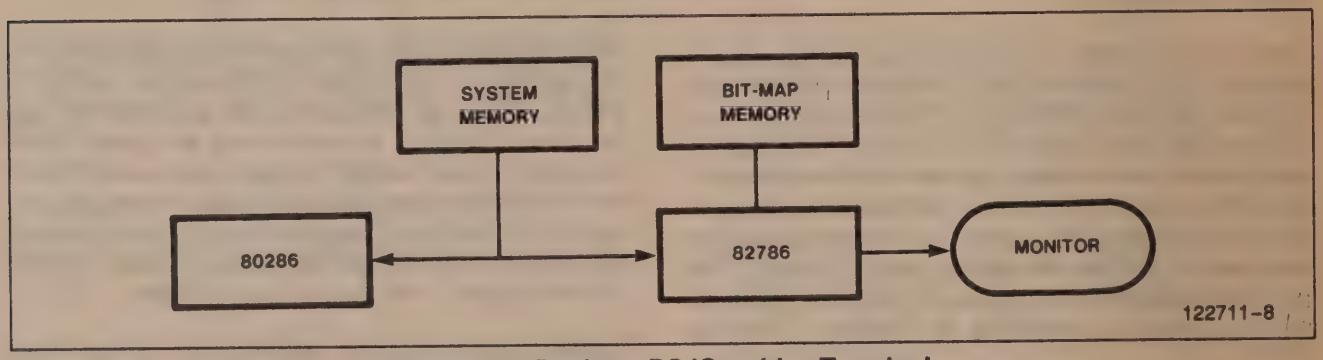


Figure 4-2. Desktop PC/Graphics Terminal

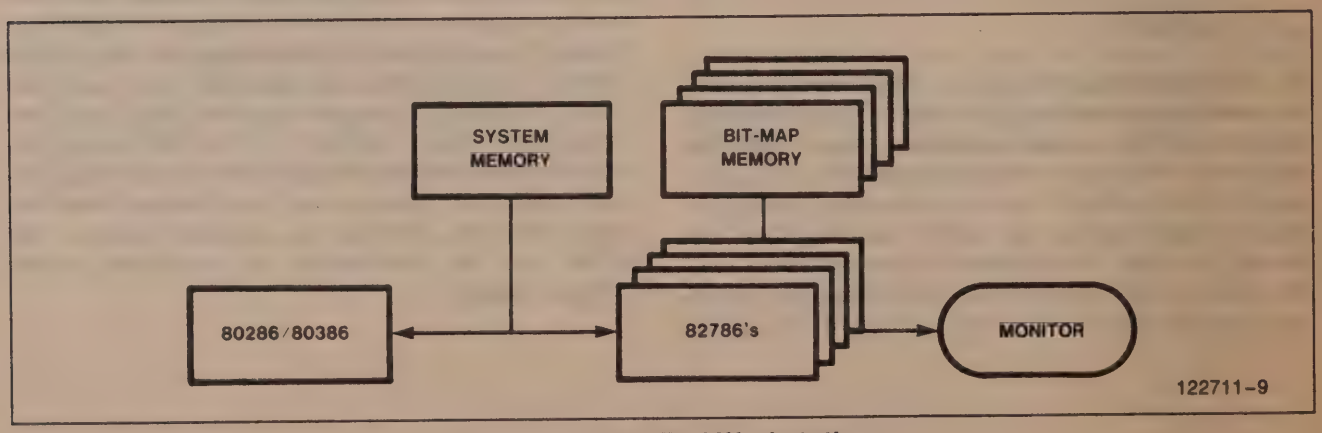


Figure 4-3. High End Workstation

tions of x1, x4, or x8. The bandwidth of the memory system can be increased by interleaving memories and/or using the Ripplemode TM or static-column mode supported by Intel CHMOS DRAMs. Both inter-leaving and Ripplemode TM are completely handled on chip and require no extra external circuits. Use of static-column DRAMs requires one 74X373 latch per bank. Interleaving refers to the use of multiple DRAM banks with one set of memories receiving new CAS signals while the other outputs data. Table 4-1 shows memory burst-bandwidth for the different configurations at 10 MHz.

DRAM refresh is done automatically by the DRAM controller. The memory array can be accessed both by 82786 internal processors (GP, DP) and by external masters (CPUs) through the BIU. The 82786 DRAM controller can be used to control system memory within its 4 megabyte address space, provided the target application can accept the decreased bandwidth of system memory. The portions of the address space dedicated to graphics and system memory are configured at initialization in the DRAM_CONTROL_REGISTER. Graphics memory is assumed to start at OH and continue up to the configuration limit. Memory addresses above this are used for system memory.

4.3 BUS INTERFACE

The Bus Interface Unit of the 82786 is designed to support all 8-, 16-, and 32-bit microprocessors from Intel, with optimization for the 80286. This permits the 82786 to run synchronously with the 80286, increasing throughput by eliminating wait states. A special 8-bit mode allows 82786 to also work with 8-bit data bus microprocessors. The 80386 itself makes interfacing to the 82786 possible. Interfacing to Intel CPUs is detailed in the Hardware Configurations Applications Note.

The bus interface allows slave access by the CPU to the graphics memory controlled through the 82786 DRAM controller. This allows the CPU to update the graphics processor instruction list and the display processor descriptor lists in the graphics memory where maximum throughput can be supported. Low-end systems could use only a single memory shared by both the 82786 and CPU and use the 82786 DRAM controller for this memory.

For performance reasons, many systems will have at least two sections of memory: the 82786 graphics memory (using the on-chip DRAM controller) and the system memory. In this configuration, the 82786 can execute bus cycles on the system bus so the 82786 can access the CPUs own memory. This master mode is designed in accordance with the 80286 definitions. This configuration allows the best of both worlds, the system and graphics memories are split for performance reasons, but the split is transparent to the software for flexibility. Character fonts and graphic objects may be retrieved from disk and placed in system memory locations reserved for access by the 82786 using a virtual mode 80286 or 80386 configuration with appropriate system software.

Table 4-1. 82786 DRAM Bandwidths

	Pagemode DRAM	Ripplemode DRAM
Non Interleaving DRAM banks	10 Megabyte/sec (diagnostics or 640 × 480 × 2)	20 Megabyte/sec (640 × 480 × 4 or 1K × 1K × 1 noninterlaced)
Interleaving DRAM banks	20 Megabyte/sec (640 × 480 × 4 or 1K × 1K × 1 noninterlaced)	40 Megabyte/sec (2K × 2K × 1 interlaced: 1K × 2K × 1. 1K × 1K × 2. 800 × 600 × 4. 640 × 480 × 8 noninterlaced)

CHAPTER 5
PACKAGE AND PIN DESCRIPTION

5.1 OVERVIEW

The 82786 is an eighty-eight pin component due to the large number of functions integrated within the device. It is available in both pin grid array and leaded chip carrier versions. The pinout of a pin grid array is shown in Figure 5-1 and a description of the pins is shown in Table 5-1.

Figure 5-1. PGA Pinout

Table 5-1. 82786 Pin Names and Descriptions

Symbol	Type	Description
A21-0	I/O	**ADDRESS LINES FOR THE LOCAL BUS:** Normally inputs for Slave Mode accesses of the 82786 supported DRAM array or internal memory or I/O mapped registers. Driven by the 82786, when it is the Local Bus Master.
D15-0	I/O	**DATA BUS:** For the 82786 DRAM array and the Local Bus.
BHE#	I/O	**BUS HIGH ENABLE:** An input of the 82786 Slave Interface: driven LOW by the 82786 when it is Local Bus Master. Determines asynchronous vs. synchronous operation for RD#, WR# and HLDA inputs at the falling (trailing) edge of RESET. A HIGH state selects synchronous operation.
RD#	I/O	**READ STROBE:** An input of the 82786 Slave Interface: driven by the 82786 when it is Local Bus Master. Asynchronous vs. synchronous input determined by state of BHE# pin at falling RESET.
WR#	I/O	**WRITE STROBE:** An input of the 82786 Slave Interface: driven by the 82786 when it is Local Bus Master. Asynchronous vs. synchronous input determined by state of BHE# pin at falling RESET.
MIO	I/O	**MEMORY / I/O INDICATION:** An input of the 82786 Slave Interface: driven HIGH by the 82786 when it is the Local Bus Master. Selects 286 Status or Command Mode vs. 8086/186 Status Mode of the 82786 Slave Interface at the falling (trailing) edge of RESET. A LOW state selects the 286 Status or Command Mode.
CS#	I	**CHIP SELECT:** Slave Interface input qualifying the access.
MEN	O	**MASTER ENABLE:** Driven HIGH when the 82786 is in control of the Local Bus, (i.e. HLDA received in response to a 82786 HREQ). Used to steer the data path and select source of bus cycle status commands.
SEN	O	**SLAVE ENABLE:** Driven HIGH when 82786 is executing a Slave bus cycle for an external master on the Local Bus. Used to enable the data path and as a READY indication to the Local Bus Master.
READY#	I	**SYNCHRONOUS INPUT:** To the 82786 when executing Local Bus cycles. Identical to 80286 timing.
HREQ	O	**HOLD REQUEST:** Driven HIGH by the 82786 when an access is being made to the Local Bus by the Display or Graphics Processors. Remains HIGH until the 82786 no longer needs the Local Bus.
HLDA	I	**HOLD ACKNOWLEDGE:** Input in response to a HREQ output. Asynchronous vs. synchronous input determined by state of BHE# pin at falling RESET.
INTR	O	**INTERRUPT:** The logical OR of a Graphics Processor and Display Processor interrupt. Cleared with an access to the BIU Interrupt Register.
CAS0#	O	**COLUMN ADDRESS STROBE 0:** Drives the CAS inputs of the even word DRAM bank if interleaved: identical to CAS1# if non-interleaved DRAM. Capable of driving 16 DRAM CAS inputs.
CAS1#	O	**COLUMN ADDRESS STROBE 1:** Drives the CAS inputs of the odd word DRAM bank if interleaved; identical to CAS0# if non-interleaved DRAM. Capable of driving 16 DRAM CAS inputs.
RAS2-0#	O	**ROW ADDRESS STROBE:** Drives the RAS input pins of up to 16 DRAMs. Drives the first three rows of both banks of DRAM.
DRA9/ RAS3#	O	**MULTIPLEXED MOST SIGNIFICANT DRAM ADDRESS LINE AND RAS3#:** Support of 1Mb DRAMs requires DRA9. When 1Mb DRAMs are used, four rows of DRAMs cannot be supported (RAS3# unnecessary) due to 82786 addressing limit of 4 Mbytes being exceeded.
WEL#	O	**WRITE ENABLE LOW BYTE:** Active LOW strobe to the lower order byte of DRAM.
WEH#	O	**WRITE ENABLE HIGH BYTE:** Active LOW strobe to the higher order byte of DRAM.

Table 5-1. 82786 Pin Names and Descriptions (Continued)

Symbol	Type	Description
DRA8–0	O	**MUTIPLEXED DRAM ADDRESS:** DRAM row and column address are multiplexed on these lines. Capable of driving 32 DRAMs without buffers.
BEN1–0#	O	**BANK ENABLE 1 AND 0:** Enables the output of the DRAM array on to the 82786 data bus (D15–0). BEN1# controls Bank 1. BEN0# controls Bank 0.
BLANK	I/O	**OUTPUT USED TO BLANK THE DISPLAY AT PARTICULAR POSITIONS ON THE SCREEN:** May also be configured as inputs to allow the 82786 to be synchronized with external sources.
VDATA7–0	O	**VIDEO DATA OUTPUT.**
VCLK	I	**VIDEO CLOCK INPUT:** used to drive the display section of the 82786. Its maximum frequency is 25 MHz.
HSYNC/ WST0	I/O	**HORIZONTAL SYNC:** Window status may be multiplexed on this pin. Can also be configured as input to allow the 82786 to be synchronized with external sources. Even as input, window status still output when BLANK is low.
VSYNC/ WST1	I/O	**VERTICAL SYNC:** Window status can be multiplexed on this pin. Can also be configured as input to allow the 82786 to be synchronized with external sources. Even as input, window status still output when BLANK is low.
RESET	I	**RESET INPUT:** internally synchronized. Halts all activity on the 82786 and brings it to defined state. The leading edge of RESET synchronizes the clock to PH1. The trailing edge latches the state of BHE# and MIO to establish the type of Slave Interface. It also latches RD# and WR# to set certain test modes.
CLK	I	**DOUBLE FREQUENCY CLOCK OUTPUT:** Clock input to which pin timings are referenced. 50% duty cycle.
V_{SS}, V_{CC}		**4 V_{SS} AND 2 V_{CC} PINS.**

A HAYDEN PUBLICATION

ElectronicDesign®

FOR ENGINEERS AND ENGINEERING MANAGERS—WORLDWIDE

FEBRUARY 17, 1983

COMPONENTS SPECIAL

• Capacitors • Precision resistors • Shielding materials
• Selecting electrolytics • Power MOSFETs for switchers

The first chip dedicated to text manipulation, the 82730 operates as a coprocessor to a host CPU and executes many high-level commands that reduce the software needed for processing text.

Text coprocessor brings quality to CRT displays

The quality of text in raster-scanning CRT displays has always been a tradeoff against the complexity, performance, and cost of the associated video system. By allocating many of the complex display functions to firmware, a dedicated text coprocessor chip, the first of its kind, replaces printed-circuit boards that contain more than 100 ICs while increasing system performance by relieving many of the host processor's text manipulation tasks. The chip thus makes possible the high quality and high performance sought, without the need to compromise because of high design complexity and high cost of text-processing hardware.

Though its speed makes the 82730 text coprocessor beneficial on its own, its utility can be enhanced considerably when working with the 82731 video interface controller. Together they provide proportional spacing, simultaneous subscript and superscript displays, dual cursors, dynamically reloadable character fonts, and user-programmable field and character attributes. By adding still another chip, the 82720 graphics display controller, the device can display high-resolution graphics and text at the same time.

Anand Balaram, Product Marketing Engineer
Andrew Volk, Project Manager
Intel Corp.
3065 Bowers Ave., Santa Clara, Calif. 95051

Housed in a 68-pin package, the 82730 text coprocessor combines a direct memory access channel and a processor bus interface that permit it to fetch its own instructions and data from the host system's memory, independent of and in parallel with the host CPU.

The two processors communicate through messages—commands, parameters, and status words—which are placed in a communication block inside a shared memory. The host issues commands by preparing messages, storing them in the communication block, and directing the coprocessor's attention to them by activating a Channel Attention signal, which is implemented in hardware. In return, the coprocessor sets a flag in the shared memory that notifies the host when it has executed the command.

The 29 high-level commands built into the 82730 break down into two groups: channel commands, which work at the system level to start and stop the display and to communicate status and similar information, and data-stream commands, which are incorporated directly into the display-data strings to govern the DMA process and control row

Copyright 1983 Hayden Publishing Co., Inc.
ORDER NUMBER: 210932

characteristics, character attributes, and so on.

The 82730 resides on a local system bus with the host microprocessor, such as the 80186 CPU, and therefore provides the same address, data, and control signals as the main processor. By handling several of the tasks typically done by the host processor—like DMA control and display formatting—it leaves the host free for other tasks.

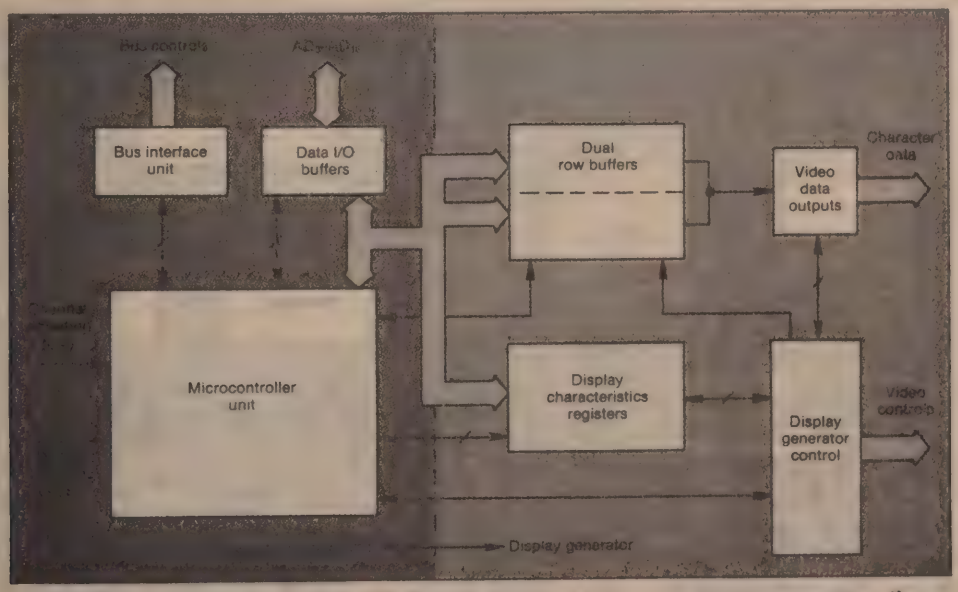
For example, when the coprocessor is configured to share the CPU bus, a portion of the host microprocessor bus bandwidth must be devoted to the DMA process that refreshes the CRT. However, the 82730's high-speed intelligent DMA controller (operating at a maximum data rate of 4 Mbytes/s) helps minimize the time spent executing the refresh operation, while simultaneously handling the formatting of the display data. A different approach involves a dual-ported memory architecture, which places the memory between the CPU and the coprocessor. That completely frees the processor bus of the refresh activity, allowing the host more time to execute other tasks. It has become a more cost-effective method, as some dynamic memory controllers now contain dual-ported arbitration logic on chip.

Inside the chip

The basic architecture of the coprocessor is divided into two main parts: a memory interface and a display generator section (Fig. 1). The memory interface lets the coprocessor and the system processor communicate via the shared memory. The display generator, in turn, responds to the data provided by the memory interface and carries out the display operations.

The memory interface actually comprises two smaller subsections, a bus interface unit and a microcontroller unit. The bus interface provides an intelligent connection from the 82730 to the host processor bus and also buffers the data transfer requests from the microcontroller. Upon initialization, the bus interface can be programmed for 8- or 16-bit data and 16- or 32-bit addresses. Furthermore, the host interface can be configured for 8- or 16-bit-wide data buses, making the coprocessor compatible with 8- or 16-bit host processors, like the 8088/80188 and the 8086/80186. Running at 8 MHz maximum in 16-bit systems, the 82730 handles the maximum DMA rate of 4 Mbytes/s.

The microcontroller unit stores the microinstructions for the 82730's high-level operations. The microcontroller's internal processor manages the memory transfers, interprets the commands embedded in the data stream, and executes those commands by sending data to the appropriate control registers or display data buffers. To optimize the transfer of data between the system and the CRT interface, the coprocessor uses three clocks—one for the host interface, the other two for video data. The memory interface section runs from the bus clock, the CRT interface operates from a reference and a character clock.



1. Divided into two main sections—a memory interface unit and a display generator—the 82730 text coprocessor can operate at optimum speed since each section can function independently at a different clock speed.

Although the coprocessor packs a considerable amount of processing power on a single NMOS chip, it cannot handle the high video dot rate needed to deliver high character counts to the CRT display. For that, it needs the 82731 video interface controller, which gains its high speed and drive capability from bipolar technology. In addition, the combination of the 82730 and 82731 succeeds in reducing the video interface to just a few latches and a software character generator residing in RAM or ROM (Fig. 2).

Inside the 82731 are the reference- and character-clock generators, a video shift register, and all attribute logic (Fig. 3). Housed in a 40-pin package, the circuit offers TTL-compatible inputs and outputs except for the video output, which is ECL-compatible and provides a dot-shift clock rate of 50 MHz maximum on characters up to 16 dots wide. The circuit proportionally spaces characters by accepting the width sent from the character generator and sending an appropriate character-clock output whose period determines the variable width of the character to be displayed.

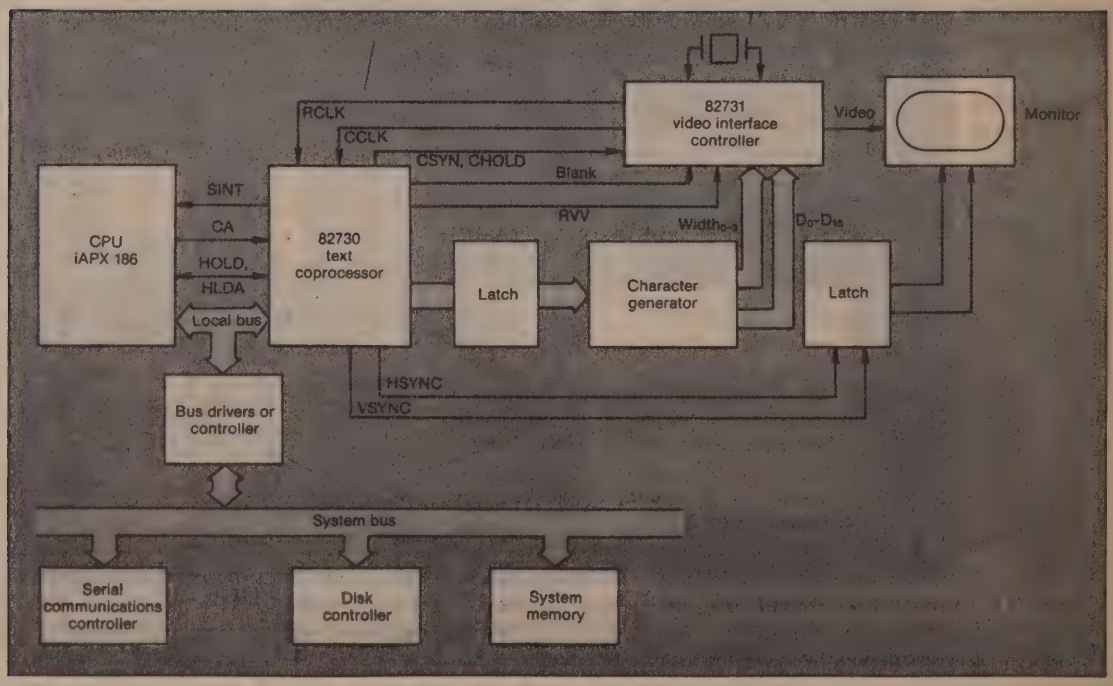
The video interface controller can employ an inexpensive, low-frequency crystal and internally multiply that frequency to generate the high-frequency dot clock. It also supports control functions such as screen reverse video, synchronized character field, and tabbing operations. The dot clock drives the internal video shift register, the character clock controls the unloading of data from the row buffers in the 82730, and the reference clock establishes the raster and screen formats. The reference clock also supplies the basic timing for the horizontal sync, blanking, border, and active display time. The corresponding vertical attributes—except border—are driven by the horizontal line time. All seven of these screen parameters are programmable by the system designer with the 82730.

System interfaces are simple

As a coprocessor, the 82730 has the same bus-control signals as an 80186 host processor and thus can share the system-bus controllers, drivers, and latches. The host processor and the 82730 arbitrate for control of the local bus through the Hold and Hold Acknowledge lines (HLD/HLDA). The Channel Attention (CA) and System Interrupt (SINT) control lines complete the wired interface. With this configuration, the 82730 has access to all the memory that the 80186 CPU has available.

Anytime the CPU wants to send a message to the 82730, it writes the command and busy flag into the 82730 command block and then pulses the coprocessor's CA input to inform it that a message is waiting. The 82730 then raises the HOLD output and waits for access to the bus. When the CPU relinquishes the bus, it raises the HLDA input of the 82730.

Once the 82730 becomes active, it transmits the command block address that was stored in its



2. A typical system built around the 82730 and the 82731 video interface controller requires very few additional ICs to mate with a host processor like the 80186. Only the system bus drivers, some latches, and a character generator are needed.

registers during initialization. That address, in conjunction with the appropriate memory control signals—including read or write strobes, lower or upper address latch enables, upper address output, or data enable output—will either read the command block or write to it. All these signals are coordinated by the bus clock.

Whenever the 82730 must send status information to the host CPU, it gains control of the bus and places the data into the status location in the command block. The bus is then released and the coprocessor notifies the CPU through the SINT signal. When the coprocessor is using a dual-ported memory to communicate with the 82730, the HOLD and HLDA signals are not employed. Instead, the 82730 accesses the dual-ported memory directly rather than acquiring the bus from the CPU.

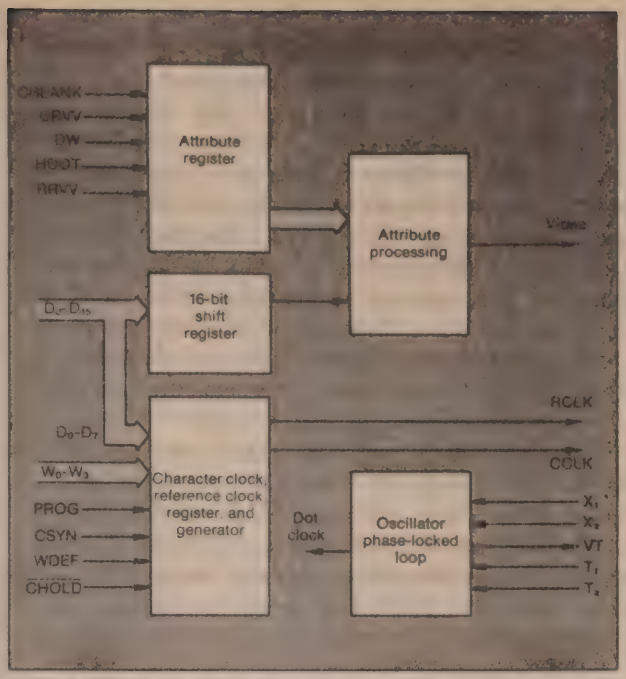
When the display process is activated, the coprocessor uses its built-in DMA capability to fetch display data from the memory. The data consists of character data mixed with data-stream commands; embedded data-stream commands provide the flexibility to manipulate data on the fly.

Soft fonts loaded

The 82730 also permits soft fonts to be automatically loaded into RAM-based character generators. Addresses and data stored in the system memory are then loaded into the row buffers of the coprocessor. During blanked rows (generally during the vertical retrace), address information is loaded into a latch and data is written to the character generator.

The 82730's dual row buffers help reduce the bandwidth and access time requirements for the system memory. The data stored in one buffer is being used to display a row on the screen while the second buffer is being loaded, by the microcontroller, with the next display row from the system memory. Up to 200 characters can be stored and displayed by each row buffer. Furthermore, since the display generator section operates asynchronously with the microcontroller unit, each can operate at optimal speed. Processing is synchronized by internal flags and shared internal storage, and data that will be displayed is exchanged through the row buffers.

The coprocessor's display generator handles the data that defines the timing and the operation of the CRT interface. That data, which is stored in the display characteristics registers of the chip, controls every aspect of the display—from the screen's format to the blink rates of the characters and cursors. All the parameters that define the initial display characteristics can be set by one command—MODEST—thus reducing the time and



3. The 82731 video interface controller is manufactured with bipolar technology, enabling it to handle video dot rates of 50 MHz and higher, which are needed by high-character-count displays. The controller serializes the parallel character outputs from the coprocessor and adds the desired attributes to each character.

effort required to establish a screen format.

Beneath the simplicity of the hardware shown in Fig. 2 are the high-level instructions—channel commands—and the data-stream commands. When combined with a table-driven linked-list data structure, they ease software development.

Central to the software is the command block, through which all channel commands are transferred between the coprocessor and the host. This block is located within the shared memory, and its exact position is set during the 82730's initialization routine (Fig. 3a). Once established, it contains all the information needed to start the display-data fetch; to communicate status, interrupt, and cursor position information; and to give the location of the mode block, which contains all the parameters for setting up the display. The START DISPLAY channel command begins the sequence (Fig. 3b).

Since the display data is set up within linked lists, the coprocessor can rapidly change any of the lists without shifting huge amounts of data. The display fetch starts with the value of the list-switch bit which selects one of two list-base pointers in the command block. The pointer points to its string pointer list; the pointers in that list direct the on-chip DMA to the data strings containing the desired display data and data-stream commands. The programmer can modify one pointer list while

displaying from the other, and can also switch screens merely by changing the list-switch bit, thus eliminating time-critical data manipulations.

Two data-stream commands—End of String (EOS) and End of Row (EOR)—are key to the linked list and DMA activities. Strings are a logical concept: they contain blocks of contiguous data stored anywhere in memory. In contrast, rows are a physical concept and represent a block of characters that make up a physical row on the screen. Many strings can exist in a display row, or many rows in a string. (Only the extra DMA overhead of fetching the new string pointer sets a practical limit on the number of strings in each row.)

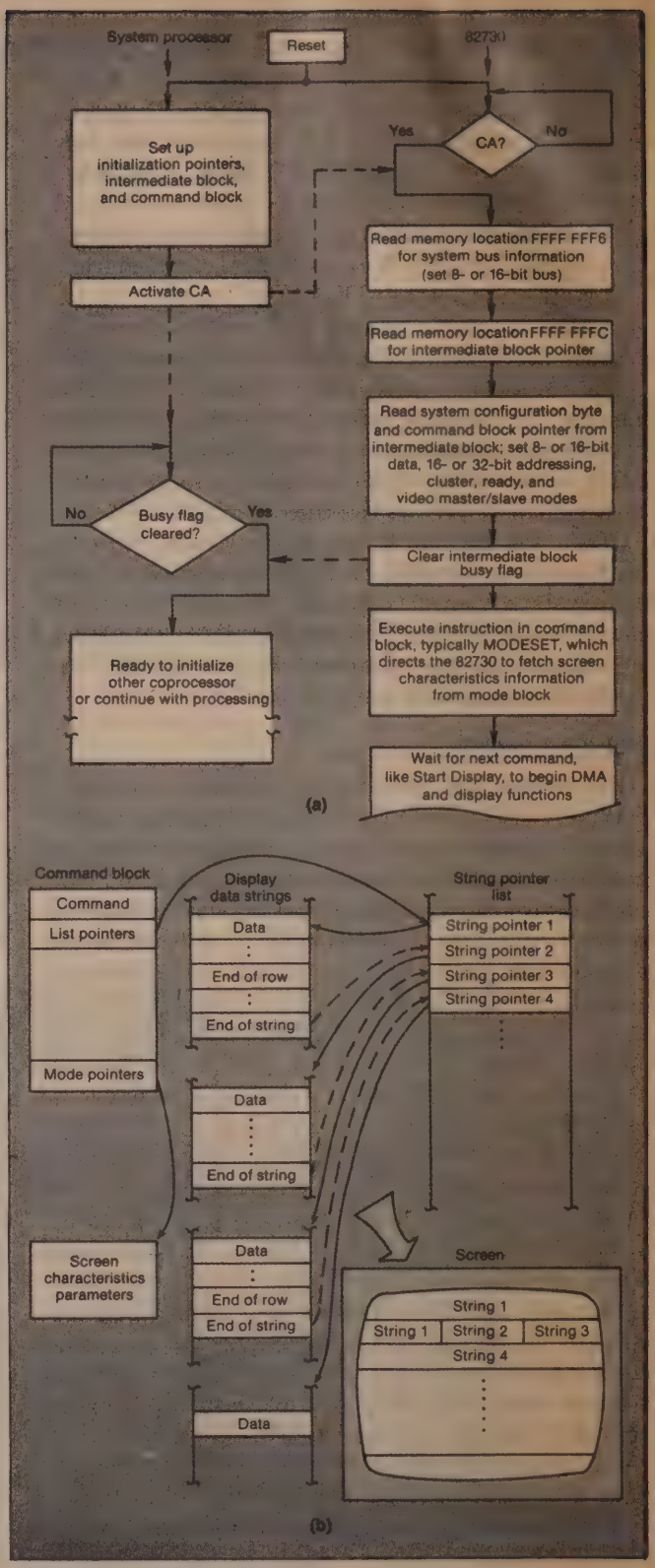
The actions of the two commands are independent. End of String tells the 82730 to get the next string pointer from the list, and from there, the next data string. End of Row suspends the DMA until the row buffers are swapped at the end of the current row. The DMA then takes over, into the new row buffer.

String manipulation fosters high speed

Strings are commonly the next level of text organization above single characters. With the 82730, a string can be as small as a character or it can be a word, row, sentence, paragraph, or a page of characters. These high-level entities can be moved merely by manipulating a small string pointer table (Fig. 5). The heart of the algorithm for word wraparound, a common feature in text processors, can easily be accommodated by a single command such as the String Compare command of the 80186. Word wraparound is then achieved by scanning the data (not moving it) and manipulating a few pointers. Earlier system designs would have required a multiple-instruction software loop that scanned and moved every individual character.

An extension of the linked list allows programmers to set up several independent data windows on the CRT screen in a virtual screen mode. That feature is especially helpful if a user wants both a menu window and one or more work-space windows. Such a scheme saves a lot of time for the end user—eliminating the back-and-forth movement between menus and working text. To set this up, several data structures, each with its own command block, can be accessed in a table-driven sequence to put data in a given window on the screen (Fig. 6).

The string list and data strings are the same for regular or virtual modes; only the structure of their command blocks differs. Thus, each virtual window can be an independent software entity in the system, and the 82730 can present these independent data bases simultaneously.



4. Both the host CPU and the coprocessor go through an initialization sequence when the computer system is reset (a). The coprocessor then looks for a START DISPLAY command so that it can load the various data strings from the system memory into the display generator section, attach attributes, and display the data on the CRT (b).

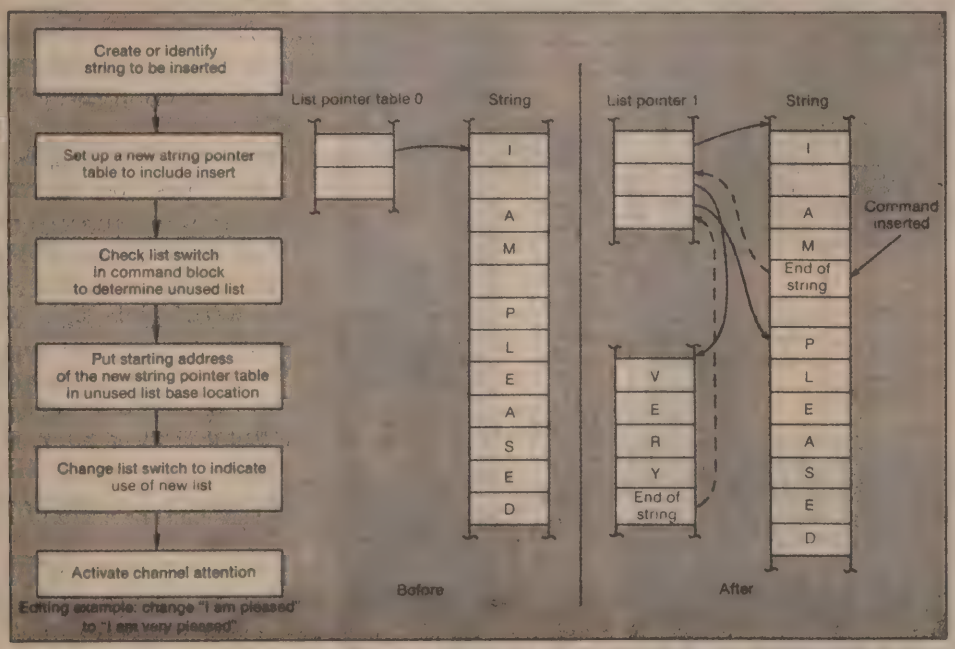
Multiple 82730s can also be used in a single system. Up to four devices can be clustered in a single system, with one serving as a system master and the others as slaves. The data for this cluster can be interleaved, permitting the cluster to work from one data base to get more characters per screen or more bits per character. Also, in the slave mode, the 82730's video outputs can be synchronized to an external video signal, giving the system such capabilities as mixed text and graphics, broadcast subtitling (text overlay), and overlays for video recording.

Attributes enhance display quality

The designers of the 82730 have given it the ability to highlight various areas of an on-screen document through the use of character and field attributes. In the 16-bit data word, for example, only the most significant bit is committed; it serves as the command or data designator. If set to 1, the word is a data-stream command, with the remaining 15 bits becoming one of the predefined instructions. However, if the MSB is 0, the other bits are at the discretion of the designer, who may choose which and how many are needed for character codes, attributes, or user-defined functions.

The 82730's six predefined attributes—reverse video, invisible, blinking character, two underlines, and a special graphics character—can be programmed to respond to any of the 15 bits, or they can be completely disabled. In addition, they can be set character by character or through a field-attribute mask. All can be attached to any character. The blinking character can be programmed for a wide range of duty cycles and blink rates. The two underlines can be independently positioned anywhere in the row height, and the position can be changed from row to row. Thus the underline can be doubled or serve as a strike-through line, a fraction line, or an overbar. One of the underlines can also be programmed to blink at the same rate as a blinking character.

The graphics character is relatively important, since it permits character information to be displayed to the full height of the row. It causes the chip's line-counter output to count from zero at the top of the display row continuously through to the bottom of the row. When used with special characters, this attribute allows business forms and graphs to be easily constructed.

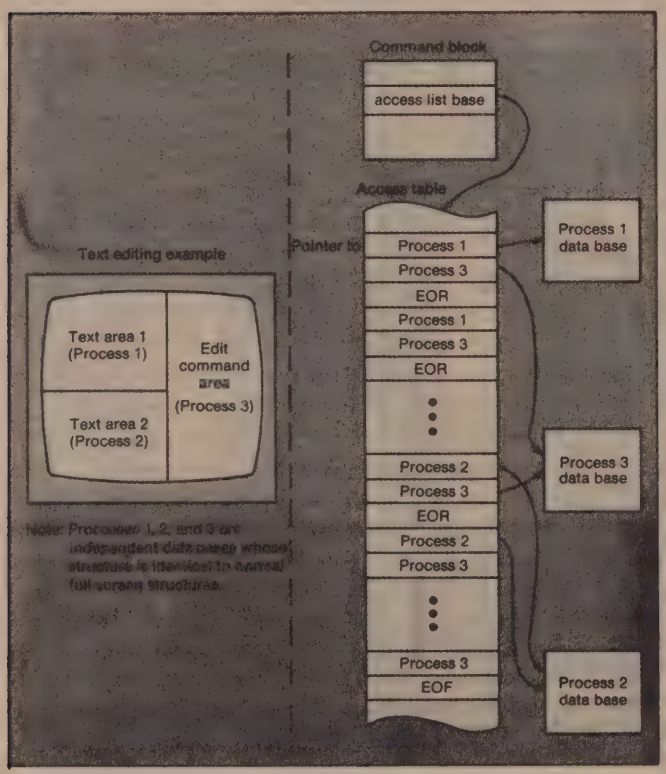


5. If a character or word must be inserted near the beginning of a screen of text, only the list pointers must be changed to add the item. In older systems, all the characters following the insertion or deletion were shifted in the memory to revise the display.

Another capability of the 82730 is subscript and superscript characters, done by manipulating the line-counter outputs. The SUB SUP N data-stream command declares which and how many pairs of characters will be displayed simultaneously as subscripts and superscripts.

Proportionally spaced displays could cause some subscript and superscript characters to have different widths and thus disrupt the vertical alignment of a character pair. A special output of the 82730 called Width Defeat prevents that misalignment by causing the 82731 video interface controller to enforce a predefined width—programmed upon system initialization—during the display of subscript and superscript characters.

The proportional spacing is performed by the reference and the character clock. Used to shift out the character and attribute data, the character clock operates during the display field. Its frequency can vary character by character, up to a rate of 10 MHz, to set the width of the character currently being displayed. The video interface controller takes the character width information that has been supplied by the character generator and produces a variable width character clock that supports the proportional spacing. This approach also greatly reduces system complexity and cost compared with previous designs.



6. The virtual window capability of the 82730 lets the user arrange independent areas in the system memory that can be displayed simultaneously on the CRT monitor.

Screen and row formats are flexible

The reference clock signal in a system that contains the 82730 and 82731 chips is a constant-frequency clock that forms the time base to generate the horizontal scan lines and vertical frame periods. One scan line can last for 256 reference clock periods, and one frame can contain up to 2048 scan lines. Within these periods, the respective synchronization pulses and the border and character fields can be set anywhere within that range. All these timing relationships, including the scan and frame periods, can be changed on a frame-by-frame basis to suit changing applications.

The screen format is flexible all the way down to the row level. For instance, the height of a row (up to 32 scan lines) and the vertical position of the characters within that row can be changed from row to row by a single data-stream command called FULROWDESCRPT. In addition, the command lets the programmer set the starting and ending scan lines within the row for the normal, subscript, and superscript character fields and the two cursors.

The same data-stream command that defines the row characteristics can also be used to blank the row, display it as reverse video, double its height (for up to 64 scan lines per row), or eliminate the proportional spacing.

Graphics, too, can be handled by the 82730, although flexibility and resolution are not as high as with the 82720 graphics display processor. Business applications typically need graphics that are no more complex than two- or three-dimensional charts or business forms. These formats can be stored as special characters in a standard font set for the character generator. Even more complex graphics can be handled through the use of mosaic graphic cells, which can be stored in RAM to permit alterations. Of course, as in most systems using floppy-disk systems for main storage, the desired fonts or graphics forms can be saved on the disks and downloaded as needed for the display.

There are many applications that also require a simple graphic display along with text—signature verification on banking terminals and general-purpose credit verification, for example.□

PROCESSING

VLSI Coprocessor Delivers High Quality Displays

Many microprocessor-based systems today use VLSI technology in processing and memory components. However, designers of subsystems have, up until now, not been able to incorporate this technology into their products because of the lack of available ICs. When, in 1981, NEC introduced the 7220 graphics display controller, users found that they could bolster system performance by off-loading graphics control chores from the system CPU. Second-sourced by Intel as the 82720, the chip uses its own drawing processor to access the required positions in the bitmap and handles both processing and display functions.

Now, Intel is poised to introduce a text coprocessor, the 82730, which is specifically tailored to execute data manipulation and display tasks. Lucio Lanza of Intel explains, "In an intelligent terminal or workstation, the CPU spends a lot of its time manipulating both graphics and text. We have identified these areas in terms of CPU use and we have distributed these blocks so that the CPU is not overburdened."

Coprocessors fall into two categories based on their architecture and operation. One type expands the microprocessor's own architecture by adding additional hardware and instructions. This type of tightly coupled coprocessor can be thought of as a transparent expansion of the microprocessor's architecture and works in sychronization with the CPU. Intel's first such coprocessor, the 8087, was designed for numerics processing and increased the microprocessor's math performance as much as 100 times.

The second type of coprocessor independently fetches its own data and sends instructions in parallel to the microprocessor. It therefore allows the microprocessor to process the tasks it handles best and delegate to the coprocessor the task it is best equipped to handle. In this cate-

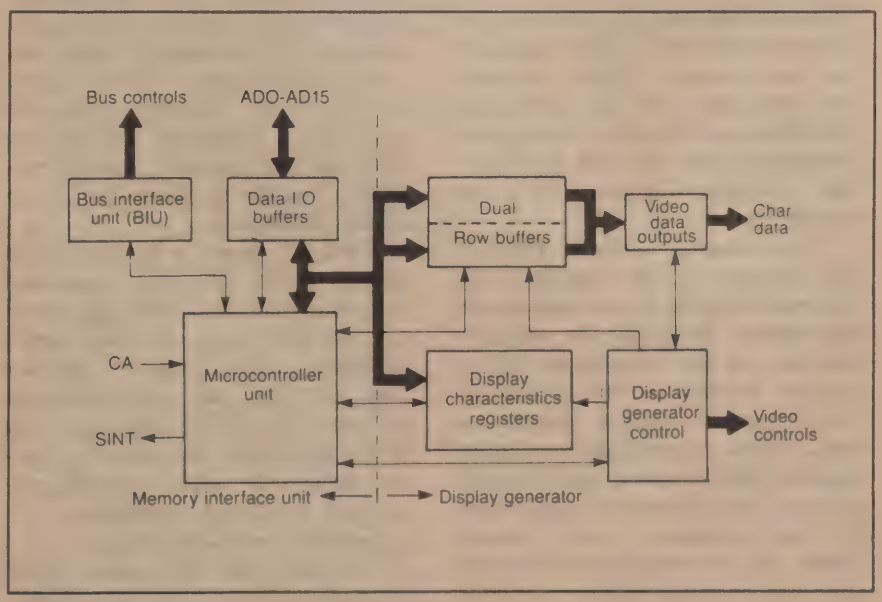


FIGURE 1: Block diagram of the 82730.

Andrew Wilson
Technical Editor

Reprinted from ELECTRONIC IMAGING © April 1983, Morgan-Grampian Publishing Company, Boston, MA 02215
50 Order Number: 231307-001

gory are I/O channel coprocessors and others that deal with communications and text processing tasks.

"The 82720 is not yet at this level," Lanza said, "since it does not have the capability of going to memory and extracting its own instruction and executing it—it needs something to spoon feed it."

Coprocessors of the second category do not monitor the CPU instruction stream. Instead, they are linked to the CPU via messages prepared and stored in shared memory. The CPU will prepare data and high level directives and then place them in memory. Upon completion of this control block, the CPU will alert the coprocessor by signaling it through a common channel attention line. From that point on, the coprocessor works on its own, fetching required data and instructions and then executing those instructions.

It is not synchronized with the CPU but works asynchronously and independently. When the coprocessor completes its task, it informs the CPU by signaling on the CPU's interrupt line.

The rationale for designing a coprocessor with one or the other architectures depends on the application requirement. Tightly coupling the coprocessor with the CPU gives the advantage of a short coprocessor preparation time but has the drawback of consuming the CPU's bus bandwidth.

In the case of numeric processing, the speed of executing the floating point algorithm is of paramount importance. Reducing the preparation time of the coprocessor task is the key because of the number of microseconds it takes to execute the task. Rapid algorithmic execution requires tight coupling. In the application of the I/O related coprocessor, the task execution time is much longer and the requirement for bus time can be much higher. And, for I/O operations the preparation time is not critical. A shared memory coupling is preferred for those types of applications because it provides greater flexibility in the design of the bus structure.

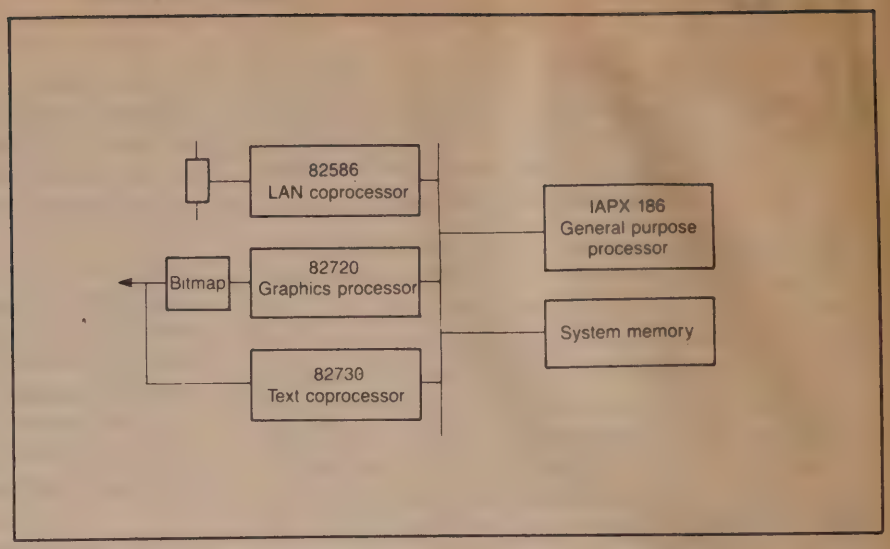


FIGURE 2: Building block approach.

Text coprocessing

"In the design of the 82730," said Lanza, "we have tried to eliminate all the known differences between what is visible on the screen and what is obtained on the printed page. In word processing systems today, even the length of a row on the CRT is sometimes not the same as the length seen in print. Clearly, when you are editing text this becomes a major problem."

The 82730 supports the generation of text displays through features which include proportional spacing, simultaneous superscript/subscript, dynamically reloadable fonts and user programmable field and character attributes. Editing capabilities are further enhanced by features such as split screen, virtual windows, smooth scrolling and table-driven linked lists.

Figure 1 shows a block diagram of the 82730. The chip is divided into two main sections—the memory interface unit and the display generator. The memory interface unit provides the communication between the 82730 and the system processor, while the display generator acts on the display data and carries out the display operation.

Communication between the 82730 and the CPU takes place through messages placed in communication blocks in shared memory. The processor issues channel commands by preparing these message blocks and directing the 82730's attention to them by activating a hardware channel attention signal (CA). The memory interface unit fetches and executes these commands. When the display process is activated, the 82730 repeatedly fetches display data and embedded datastream commands from memory utilizing its built-in DMA capability, executes any datastream commands as encountered on the fly, and loads the row buffers with the display data. After executing these commands, the 82730 clears a busy flag in memory, to inform the host CPU that it is ready for the next command.

The memory interface unit is divided into two sections—the bus unit and the microcontroller unit. The bus interface unit provides the electrical interface to the system bus and the timing signals required for the microcontroller unit operations, making these operations transparent to the microcontroller unit. The 82730 can be programmed during initialization to provide 8 or 16 bit data, and 16 or 32 bit addressing.

The microcontroller unit contains the microinstruction store and the associated circuitry required for the execution of all channel and datastream commands. It uses the bus interface unit in carrying out its memory access tasks such as loading the row buffers with display data.

The interaction between the microcontroller unit and the display generator takes place through shared internal storage. The microcon-

"The device provides the ability to independently maximize the performance of the CPU."

troller unit fetches data from memory and writes it in the internal storage, while the display generator reads from the internal storage and carries out the display operation. The microcontroller unit and display generator operate asynchronously with respect to each other. Synchronization is accomplished through communication via internal flags and display timing signals generated by the display generator. The internal shared storage consists of row buffers which store the display data and internal registers which store display parameters. There are two row buffers each capable of storing up to 200 characters. The data in one row buffer is used by the display generator to display one complete character row on the screen, while the microcontoller unit is loading the second row buffer with display data fetched from memory. At the end of the row being displayed, the buffers are swapped and the microcontroller unit and display generator resume their tasks.

The display characteristics registers contain all the information used to control every aspect of display characteristics from screen size to blink rates. A major portion of this register set is the three content addressable memory (CAM) arrays that allow flexible timing control for row and screen characteristics. The user has the power to set the parameters for the entire screen by invoking a single high-level command.

By separating the video interface clocks from the bus interface clock, the 82730 provides the designer with the ability to independently maximize the performance of the CPU and video sections of the system.

The video interface consists of two independent clocks: the Reference Clock (RCLK) and the Character Clock (CCLK). While the RCLK controls the raster timing and defines the screen layout, the CCLK independently shifts character and attribute information out of the 82730, which allows proportional spacing to be achieved.

Combining text and graphics

A major requirement in the design of engineering workstations is the simultaneous display of both text and graphics. In terms of graphics requirements, the designer of such systems needs a drawing processor for fast geometric primitives, a math processor for fast transformations and a general purpose processor for access to the graphics database.

For text, string processing is needed for manipulation of text primitives and database processing is needed for access to the document files. The solution to this problem can be solved by using both the 720 graphics coprocessor and the 730 text coprocessor (Figure 2).

Both coprocessors work with Intel's new 82586 communications coprocessor. This works in conjunction with a CPU and the appropriate software to provide local area network (LAN) control capabilities. Message data to be placed on the network by a microprocessor-based work station is stored in shared memory in transmit blocks. Pointers (starting address information) to these blocks are stored along with processing instructions in other shared memory blocks. Status information and overall directives are stored in system control blocks which serve as the mailbox between the CPU and the 82586.

When alerted by a channel attention signal, the 82586 will perform a host of tasks involved in accessing data to be transmitted from its location in memory, framing the message packets containing the data and seeing to the transmission on the network medium. In addition, the 82586 receives and buffers incoming data which it then stores in shared memory for the CPU to collect. It is the CPU's job to allocate the blocks of memory for the LAN coprocessor to store the received packet data. ■

intel

ARTICLE REPRINT

AR-296

September 1983

MIGHTY CHIPS

By **James Fawcette**
PC World Vol 1, #5

"Reprinted by permission of PC World from Volume 1, Issue 5, published at 555 De Haro Street, San Francisco, CA 94107."

"Subscription rates $24/yr. PC World Circulation Department, PO Box 6700, Bergenfield, NJ 07621

Order Number 230810-001

Something exciting is going on. But like most significant events, it is not happening quickly. Spurred on by developments in integrated circuit technology, a new generation of personal computers is taking shape, and the IBM PC and its clones are at the forefront.

As IBM PC users, it's sometimes hard to remember that the inanimate metal boxes in front of us are susceptible to evolution. But occasionally a product is introduced that forces the complete redesign of our personal computers.

Integrated circuits (ICs), the devices that bring intelligence to our machines, have reached a new level of technological achievement, and now the computers that use them must advance as well. Strange as it seems, these small silicon chips are setting the guidelines for the next generation of personal computers.

THE CHIP MAKERS

Now that personal computers have caught on, the semiconductor manufacturers who make ICs are eyeing the swelling market for personal computer ICs.

Dozens of newly developed semiconductor chips are being aimed at the personal computer market. These chips range from hard disk controllers that speed access time to linear predictive coding processors for speech recognition. With these new ICs driving personal computer design, we'll soon see machines we once only reasoned would exist: diskless computers running a wide array of software loaded over telephone lines; computers that display text exactly as it will be printed, with justified margins, superscripts and subscripts, and bold and italic typefaces on screen; and systems with greater, more accessible graphics.

As computer design is simplified by these advanced ICs, product differentiation will become greater. This portends the death of those PC clones capable only of basic spreadsheet and word processing operations. Instead, to survive in the increasingly cost-competitive, standardized personal computer market, small-system manufacturers will tailor their products for niche markets.

BIG BLUE

Intel Corporation, located in Northern California's renowned Silicon Valley, is one of the largest and most innovative chip manufacturers in the industry. IBM has been committed to Intel products for years; the PC is built around Intel's 8088 microprocessor and, as recently as late 1982, IBM invested $225 million in a minority share of Intel stock. A commitment this size is a good indicator of IBM's faith in Intel products. IBM's good faith and multimillion-dollar investment is guaranteed by Intel's long-standing promise that software written for the 8088 will run on all its future processors.

By taking a close look at the Intel ICs, we can gain valuable insight into the capabilities of the IBM PCs that will be built around them. The design philosophy of Intel's IC family differs radically from that of competitors Motorola, National Semiconductor, and Zilog. Diverse chip designs mean that the system designs of the IBM PC and its competitors, such as Apple's Lisa (based on the Motorola 6800 microprocessor), will also be radically different.

THE MICROPROCESSORS

Of the many Intel chips being produced, some will have a greater impact on the computer industry than others. In the vanguard will be the new microprocessors.

Design of the PC was shaped by IBM's surprising selection of the 8088. This choice caught most industry observers off guard since IBM, also the world's largest semiconductor manufacturer, had traditionally used its own designs for computer logic. Once Big Blue settled on the 8088, Intel's design philosophy was firmly implanted in the PC—from the 8088's segmented memory scheme to its 16-bit registers and 8-bit bus.

Like the 8088, each of the four microprocessors Intel is now readying for production could dramatically influence the design and performance of tomorrow's PCs.

The 80186. The recipe for putting an entire central processing unit (CPU) board on one chip is easy. Take an; 8086 (the 16-bit bus big brother of the 8088), speed it up, and then add most of the support chips essential to making the 8086 run in a personal computer. Reduce the size with the help of computer-aided design until all the chips fit onto one sliver of silicon, and *voila,* you have the 80186 (186), an entire motherboard on a chip.

While firming up plans for full-scale production of the 186, Intel is currently providing samples of the chip to computer manufacturers, including MAD Computer and Durango Systems. The rewards for using this newest chip are many: manufacturing costs are cut since a single IC is less expensive to buy than a boardful of them; physical CPU size is reduced, opening the way to shrink overall computer size or to put more power in the same box; and development time is cut for computer designers, which means considerable savings for system makers.

The 80188. If the 186 is too rich for your taste, the 80188 (188) may be more suitable. As with the 186; the 188's core CPU and support chips are melded on a single IC; like the 8088, however, the 188 has an 8-bit interface to the outside world (the 186 has a 16-bit interface). The 188 decreases costs by allowing computer manufacturers to use less expensive 8-bit peripherals. Although the 186 has received more publicity so far, the 188, aimed squarely at the massive 8-bit computer market, is expected to be used in greater numbers, at least in the short term.

The 80286. Powerful multiuser systems will benefit the most from the 80286 (286), possibly the most powerful microprocesor commercially available to date. Squeezing 150,000 transistors on a chip, the 286's designers have integrated a pair of HMOS-III (Intel's own proprietary process technology) 8086s and numerous other very large scale integration (VLSI) components. The resultant chip is two to three times faster than the Motorola 68000 even though both chips can address about the same amount of

memory. The 286 has very high speed (1.5 million instructions per second, five to six times faster than the 8086), about 16 megabytes worth of addressable physical memory, the ability to address a virtual memory of 1 gigabyte per task (equal to the capacity of 100 IBM XT Winchester drives), and the ability to provide several layers of muiltiuser security on chip.

The 80386. Not yet built, the 80386 (386) is promised for 1984, but the release date may slide to 1985. If the 286 is vastly more powerful than the 8088 or 8086, then the 386's potential is staggering. Complementary metallic oxide semiconductor (CMOS) process technology, which lowers power consumption, is being used to build this 32-bit chip. Intel, Motorola, and National Semiconductor are already jockeying for position in what will be an intense competition for the 32-bit market. Motorola is claiming that its 68020 will be the first widely available 32-bit microprocessor when it is introduced later this year, although NCR has already scooped the industry with its 32-bit chip. Hewlett-Packard, not to be outdone, has put 450,000 transistors on a single proprietary 32-bit microprocessor, which is used in the $20,000 to $30,000 HP 9000 work station.

How will these processors impact the personal computers that use them? The most obvious effect will be faster performance. Even the budget model 188 boasts two to five times the instruction and execution speed of the 8088 in today's PC. A 286 is about twice again as fast as the 188, and next year's data-gobbling 386 will have more speed than anyone can immediately use.

Since the 188 is ideal for low-priced portable computers, it ceates the ironic probability that a PC-compatible portable may soon by available that will run the IBM PC's full line of software and run it faster than the full-sized PC.

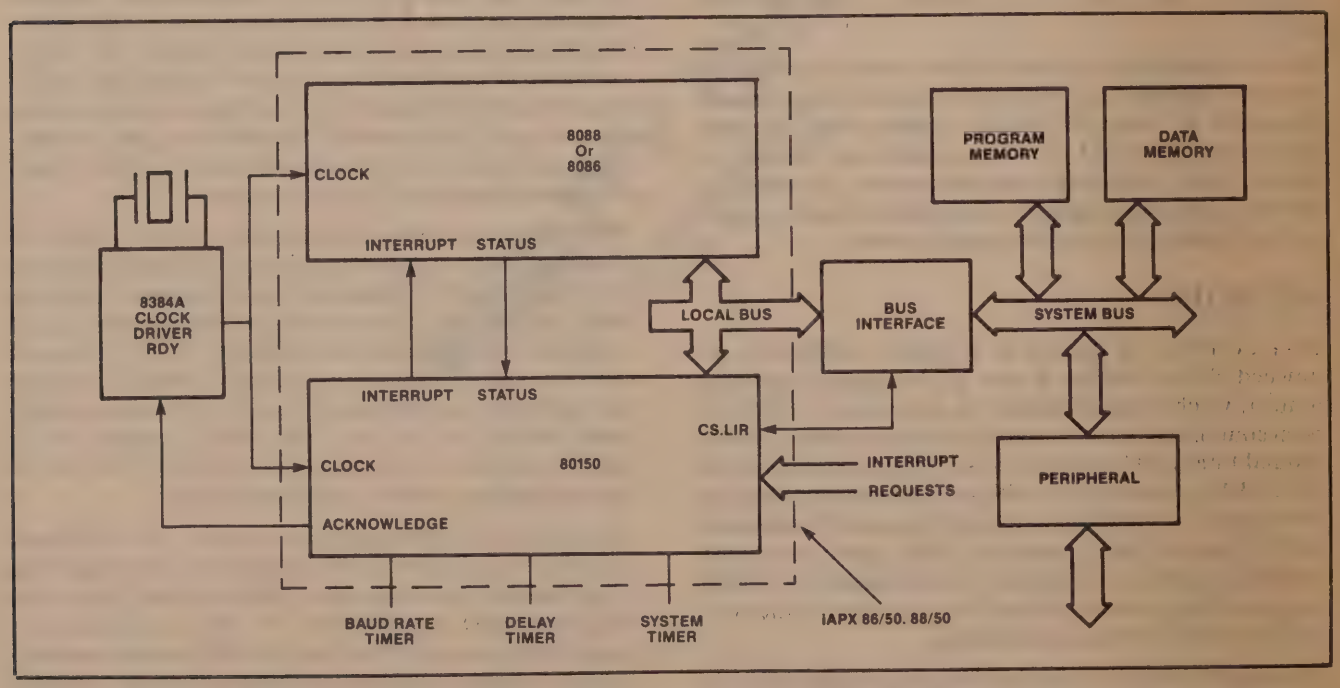
SOFTWARE ON SILICON

One chip ready to plug into the next generation of personal computers is the 80150 (150) CP/M software-in-silicon operating system. A complete CP/M-86 operating system is stored in ROM on this chip, along with drivers for input and output devices.

Use of a 150 CP/M chip will eliminate the traditional booting up procedure of loading an operating system disk and reading its contents into operating RAM. Instead, the user will simply turn on the computer and press a CP/M-86 button. Again and more importantly, this chip lowers overall computer production costs since a disk drive and attendant control circuits are replaced by a solitary chip.

Another chip, similar to the 150, has Intel's proprietary RMX operating system in silicon. This little-known RMX chip is also suitable for present and future IBM PCs.

Many people question the wisdom of putting software in silicon. "Software should be soft," says Bill Gates, chairman of the board at Microsoft. He points out that operating systems are constantly updated; for instance, Microsoft will soon offer a revised version of MS-DOS that supports networking. Such updates can't readily be added to a hardware production line and certainly won't help the ROM chips already in users' computers.



BLOCK DIAGRAM OF INTEL'S 80150 CP/M ON A CHIP WITH THE 8088 OR 8086 MICROPROCESSOR

Still, Intel argues that its choice of CP/M makes the 150 practical. "We picked CP/M because it is a mature operating system," Says Intel's product marketng engineer for software on silicon, Carl Buck. "We'd have more difficulty with a less developed product." The many versions of MS-DOS helped eliminate that operating system from consideration. Yet according to Digital Research President John Rowley, Intel left some room on the 150 chip to add to CP/M in the future.

Also, use of the 150 CP/M chip doesn't preclude the use of other operating systems. PC-DOS could still be loaded into a system and run, making use of the 150's input/output drivers.

PORTABLES

Having software on silicon opens the way for very powerful diskless portable computers. The minimum configuration for a 188-based unit with the 150 CP/M operating system could include one or two BASIC applications programs in ROM, providing spreadsheet and word processing power in a unit the size of a keyboard with a small flip-up screen. Intel Product Marketing Engineer Tony Zingale suggests we may soon see truly usable portables selling for around $500.

More ambitious and expensive portables could accept applications software over telephone lines, loading them into a variety of media. Several memory technologies will compete for room in portable computers, including magnetic bubble memories, already being used in the Grid and Teleram computers. Commercially available bubbles have 4 megabit capacity, while 10- to 16-megabit bubbles are projected for the near future. Japan's NEC reported a major breakthrough that within 5 years will allow bubbles to store 1 gigabit of data. Of course, 8 of those bits are needed to store 1 byte of data.

Vying with bubbles in some applications and complementing them in others are electronically programmable read-only memories, or EPROMs. Like ROM, EPROMs are nonvolatile chips. Unlike ROM, EPROMs can be reprogrammed. Intel now offers 256K EPROMs, and it is anticipated that other companies will offer 256K EPROMs before the year's end.

GRAPHICS

The space created on the motherboard by the 186 and friends will enable computer designers to add more graphics capability to their systems. Like the 150 there are co-processor chips ready for the task.

A pair of Intel ICs, the 82720 (720) graphics display controller and the 82730 (730) text co-processor, are touted as providing vastly enhanced and simplified displays. With the 730, text can be displayed on the computer screen as it will be printed out. Italics can be mixed with straight text, and superscripts and subscripts are shown without the annoying and often misleading arrows common in today's software.

Editing can be speeded up by the 730's support for split screens, multiple windows, dual cursors, smooth scrolling, and table-driven linked lists. Displays of up to 200 characters per row and 128 lines per screen can be supported, and unique character sets, such as Arabic or Japanese, can be built.

Even more capability can be added though the 720, an IC that works with or without the 730. Introduced in September 1982, the 720, a joint effort between Intel and NEC, is said to be integral to graphics plans for NEC's 8086-based Advanced Personal Computer.

One application in which the 720 and 730 will shine is opening windows on-screen. Most computer users are familiar with the ability of Apple's Lisa to link spreadsheets, graphics, and word processing through multiple displays, or windows, on one screen. Lisa uses memory-hungry software and dedicated hardware. Apple's initial release uses 1 full megabyte of RAM, and Lisa will soon be offered with 4M of internal memory in addition to a mandatory 5M hard disk.

For comparison, the IBM PC, limited by the range of the 8088, can address 1M tops. VisiCorp's *Visi/ON* promises Lisa-like graphics and program-linking capabilities for the IBM PC, with lower memory demands and no dedicated hardware other than a mouse. Although *Visa/ON* supposedly runs faster with an 8087 math co-processor, VisiCorp will not comment on whether its software will make use of the 720 or the 730.

BIT-MAPPED GRAPHICS

Both Lisa and *Visa/ON* use bit-mapping, a process that the 720 and the 730 are said to simplify. In plain words, to create an image on-screen, the electron gun that illuminates the screen must be positioned and then turned on and off. Data to do this is stored in RAM as a bit-map memory corresponding to positions of pixels lit on the screen. For one-level monochrome displays, 1 memory bit describes each pixel; for color and levels of grey, several bits must be used to describe each pixel.

Creating images is a lengthy chain of simple operations. In a system that uses the 8088 alone, the microprocessor is heavily burdened and the software runs slowly. Using complementary chips to take up part of the processing chore will speed up the process considerably. This is where the 720 and the 730 come in By doing tasks such as looking up and manipulating a library of commonly used figures, quickly accessing the bit-map memory, and rewriting the bit map, both chips speed text and graphics operations.

FLAT VS. SEGMENTED MEMORY

Use of the 720 and the 730 demonstrates Intel's design philosophy and how this philosophy impacts the IBM PC. Computers such as Lisa that are based on the Motorola 68000 have a flat memory, while computers based on the 8088 or 8086 use segmented memory. According to Intel,

segmented memory (see "How the PC thinks," *PC World,* Vol. 1, No. 1) works better for text and graphics manipulation than its flat counterpart. Ordinarily in processing any string of characters, changing a single letter in a string of text means repositioning every character in a document. But since segmentation uses pointers to locate data in memory, only the pointers locating the beginning and the end of a passage of text have to be changed. Similarly, pointers in memory can be used to position bit-map data corresponding to multiple windows on-screen, eliminating the need to recalculate and manipulate the entire bit map. Segmented vs. flat memory has become somewhat of a religious issue in the semiconductor industry.

Intel and Motorola also differ on how much burden to put on the CPU. Motorola's 68000 is faster than the 8088 and the 8086 and can address more memory than either of those chips or the 188 or the 186. But the 186 and the 286 are substantially faster than the 68000. Also, the 286's ability to address 16M opens the way to using large memory segments, strengthening Intel's case for segmented memory.

In many 68000-based high-end systems the computer designers have decided to use a co-processor, either bit slice, or in one case, an 8086, to do graphics. Many people are skeptical of Intel's graphics approach, but Intel maintains that its approach will allow computer designers greater flexibility. In an ultimate system, multiple 720s and 730s could be combined to handle interactive windows under the direction of a 286 processor, while more complex imagery (beyond the practical ability of bit-mapping) could be managed by an 80287 math co-processor, the next generation cousin of the 8087. The creation of three-dimensional graphics that can be rotated on screen for advanced computer-aided design and manufacturing systems, for instance, is best handled by Vector Graphics rather than bit-mapping.

SOFTWARE DEMANDS

Yet there is more to computer design than hardware. Software must be written to take advantage of the new IC's promise. In the case of the 286, no operating system yet exists that can take full advantage of its operating capabilities. Plug-ins currently on the market that add the 286 to the IBM PC provide little more than a faster 8086. Only new operating software will use the new chips to their fullest potential.

One solution on the horizon is a 286 version of XENIX due to be introduced mid-1983. XENIX, a multiuser operating system with a visual shell similar to MS-DOS, is a takeoff on Bell Labs' UNIX operating system. A licensing agreement among Intel, Bell Labs, and Microsoft, the author of XENIX and MS-DOS, is reported being negotiated. Negotiations between Intel and Digital Research to provide a CP/M variant for the 286 have been underway for some time but have reportedly stagnated.

For lower-end systems Microsoft is said to be upgrading MS-DOS to accommodate networking. This advance comes at the right time, as the 188 and 186 open up sockets that could be used for local area network chips such as the programmable Ethernet chip from Intel.

As long as software and hardware keep growing rapidly together, PC users will be offered a continuing stream of improved computers and ever more capable plug-in boards. The variety seems endless and next year's crop exciting.

Eraseable/Programmable Logic Devices

8

5C121
1200 GATE CHMOS
H-SERIES ERASABLE PROGRAMMABLE LOGIC DEVICE

- **High Performance LSI Semi-Custom Logic Replacement for Gate Arrays and Conventional Fixed Logic**
- **EPROM Technology Based. UV Erasable**
- **Programmable Macrocell and I/O Architecture; up to 36 Inputs or 24 Outputs, 28 Macrocells Including 4 Buried Registers**
- **All Inputs are Latchable with a Programmable Latch Feature**
- **High Speed t_{PD} (Max) 50 ns Operating Frequency (Max) 15 MHz**
- **Low Power; 15 mW Typical Standby Dissipation**
- **Typical Usable Gate Count of 1200 2-Input NAND Gates**
- **Advanced Architecture Features Including Programmable Output Polarity (Active High/Low), Register By-Pass and Reset Controls**
- **Programmable Clock System for Input Latches and Output Registers**
- **Product-Term Sharing and Local Bus Architecture for Optimized Array Performance**
- **Compatible with LS TTL and 74HC CMOS Logic**
- **Register Pre-Load and Erasable Array for 100% Generic Testability**
- **Programmable "Security Bit" allows total protection of proprietary designs**
- **Available in a 40-Lead Window Cerdip Package** (See Packaging Spec, Order #231369)

The Intel 5C121 H-EPLD (H-series Erasable Programmable Logic Device) is an LSI logic circuit that is user customizable through programming. This device can be used to replace gate arrays, multiple programmable logic arrays and LS TTL and 74HC CMOS SSI and MSI logic devices. The logic capacity of the 5C121 is typically equal to 1200 two-input NAND gates.

The 5C121 H-EPLD uses CHMOS* EPROM (floating gate) cells as logic control elements instead of fuses. Use of Intel's advanced CHMOS II-E EPROM process technology enables greater logic densities to be achieved with superior speed and power performance. The EPROM technology also enables these devices to be 100% factory tested by the programming and the erasure of all the EPROM logic control elements in the device.

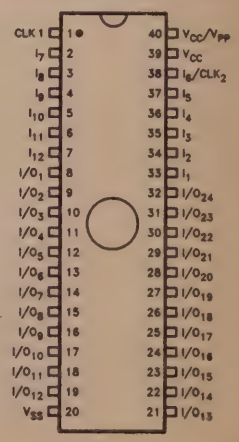
The architecture of the 5C121 is based on the 'Sum of Products' PLA (Programmable Logic Array) structure with a programmable AND array feeding into a fixed OR array. Flexibility in accommodating logical functions without the overhead of unnecessary product terms or speed penalties of programmable OR structures is achieved through the provision of a range of OR gate widths as well as through product term sharing. The use of a segmented PLA structure with local and global connectivity allows for further improvements in performance. The 5C121 also contains innovative architectural features that provide extensive Input/Output flexibility.

*CHMOS is a patented process of Intel Corporation.

RECOMMENDED OPERATING CONDITIONS

Symbol	Parameter	Min	Max	Units
V_{CC}	Supply Voltage	4.75	5.25	V
V_I	INPUT Voltage	0	V_{CC}	V
V_O	OUTPUT Voltage	0	V_{CC}	V
T_A	Operating Temperature	0	70	°C
t_R	INPUT rise Time		500	ns
t_F	INPUT fall Time		500	ns

Pin Configuration



290098-1

ILLUSTRATIONS COURTESY OF ALTERA CORPORATION.

Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied. Information contained herein supersedes previously published specifications on these devices from Intel. **September 1985**
© Intel Corporation, 1985

Order Number: 290098-001

ARCHITECTURE DESCRIPTION

The 5C121 H-EPLD has 12 dedicated inputs as well as 24 Input/Output pins. All inputs to the circuit (both dedicated and I/O inputs) may be latched using transparent 7475 type latches. In addition to these 36 input latches, 28 D type registers are also provided.

The internal architecture of the 5C121 H-EPLD is based on 28 macrocells. Each macrocell (see Figure 1) contains a PLA structure (programmable AND array product terms connected to an OR gate) and an I/O architecture control block (with a D flip flop) that can be programmed to create many different output logic structures. This powerful I/O architecture can be configured to support both active-high, active-low, 3-state, open drain and bi-directional data ports all on a 4-bit wide basis. They can also act as inputs on a nibble wide basis with optional input latching.

Macrocells in each half of the circuit are grouped together for I/O architecture programming. Each bank of four macrocells can be further programmed on an individual macrocell basis to generate active high or active low outputs of the logic function from the PLA.

The primary logic array of the 5C121 is segmented into two symmetrical halves that communicate via global bus signals. The main array contains some 15104 programmable elements representing 236 product terms (AND gates) each containing 64 input signals.

The macrocells share a common programmable clock system (described in a later section) that controls clocking of all registers and input latches. The device contains 8 modes of clock operation that allow logic transition to take place on either rising or falling edges of the clock signals.

The circuit further contains four macrocells whose outputs are not tied to any I/O pin but feedback into the array to create buried state-functions. The feedback path may be either the registered or combinational result of the PLA output. The use of the buried state macrocells provides maximum equivalent logic density without demanding higher pin-count packages that consume valuable board space.

MACROCELL I/O ARCHITECTURE

The Input/Output architecture of the 5C121 macrocell (see Figure 1) can be programmed using both static and dynamic controls. The static controls remain fixed after the device is programmed whereas the dynamic controls may change state as a result of the signals applied to the device.

The static controls set the inversion logic (i), register by-pass (ii) and input feedback multiplexers (iii). In the latter two cases these controls operate on four macrocells as a bank.

The buried-state registers have simpler controls which determine if the feedback is to be registered or combinational.

The inversion control logic, marked (i) in Figure 1, is achieved by programming the EPROM control bit connected to the same XOR gate as the output from the PLA structure. Programming or erasure of this EPROM element toggles the OR gate output of the PLA between active-high and active-low. The inversion control operates on an individual macrocell basis.

The register by-pass control, marked (ii) in Figure 1 allows the PLA output to either flow through the D flip flop as a registered output or by-pass the flip flop and be a combinational output.

The dynamic controls consist of a programmable input latch-enable as well as reset and output enable product terms. The latch-enable function is common throughout the 5C121 and once chosen this function will latch all the inputs. This function is programmed by the clock control block but may also be driven by input signals applied to pin 1 (see clock modes—Table 1).

The reset and output-enable controls are logically controlled by single product terms (the logic AND of programmed variables in the array). These terms have control over banks of four macrocells.

The output-enable control may be used to generate architecture types that include bi-directional, 3-state, open drain or input only structures.

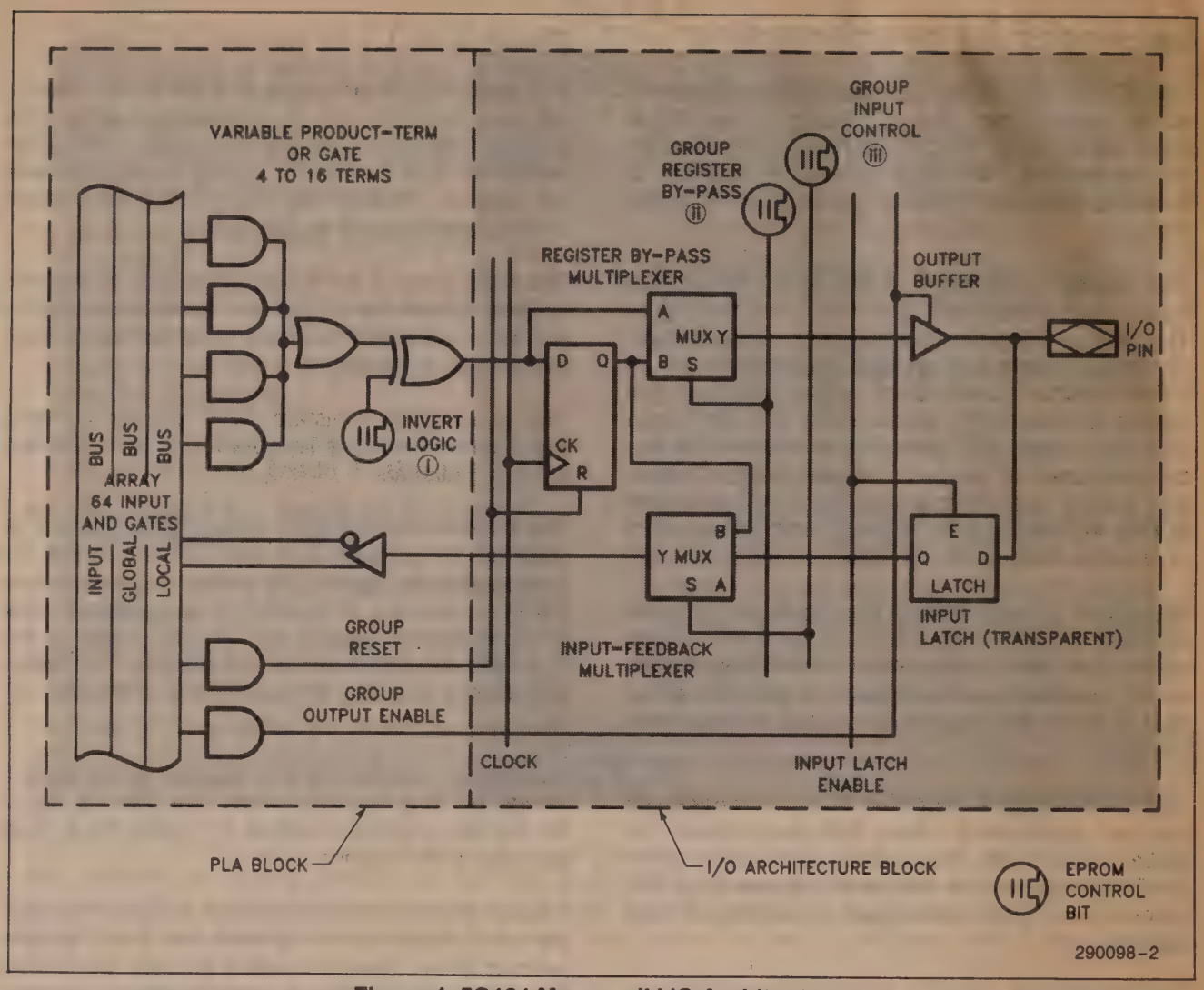


Figure 1. 5C121 Macrocell I/O Architecture

INTERNAL BUS STRUCTURE

The two identical halves of the 5C121 communicate via a series of busses. The local bus structure that is used for communication within each half of the chip contains 16 conductors that carry the TRUE and COMPLEMENT of 8 local macrocells. In the block diagram (Figure 2) of the 5C121 the local macrocells are B-1 and B-2 on one half and A-1 and A-2 on the other half.

The global busses (Input bus & Global feedback from A-3 & B-3 macrocells & buried registers) are made up of 48 conductors that span the entire chip. These 48 conductors carry the TRUE and COMPLEMENT of the twelve primary inputs (pins 2 through 7 and 33 through 38), signals from 4 Buried Registers as well as the global outputs of 8 macrocells in groups A-3 and B-3.

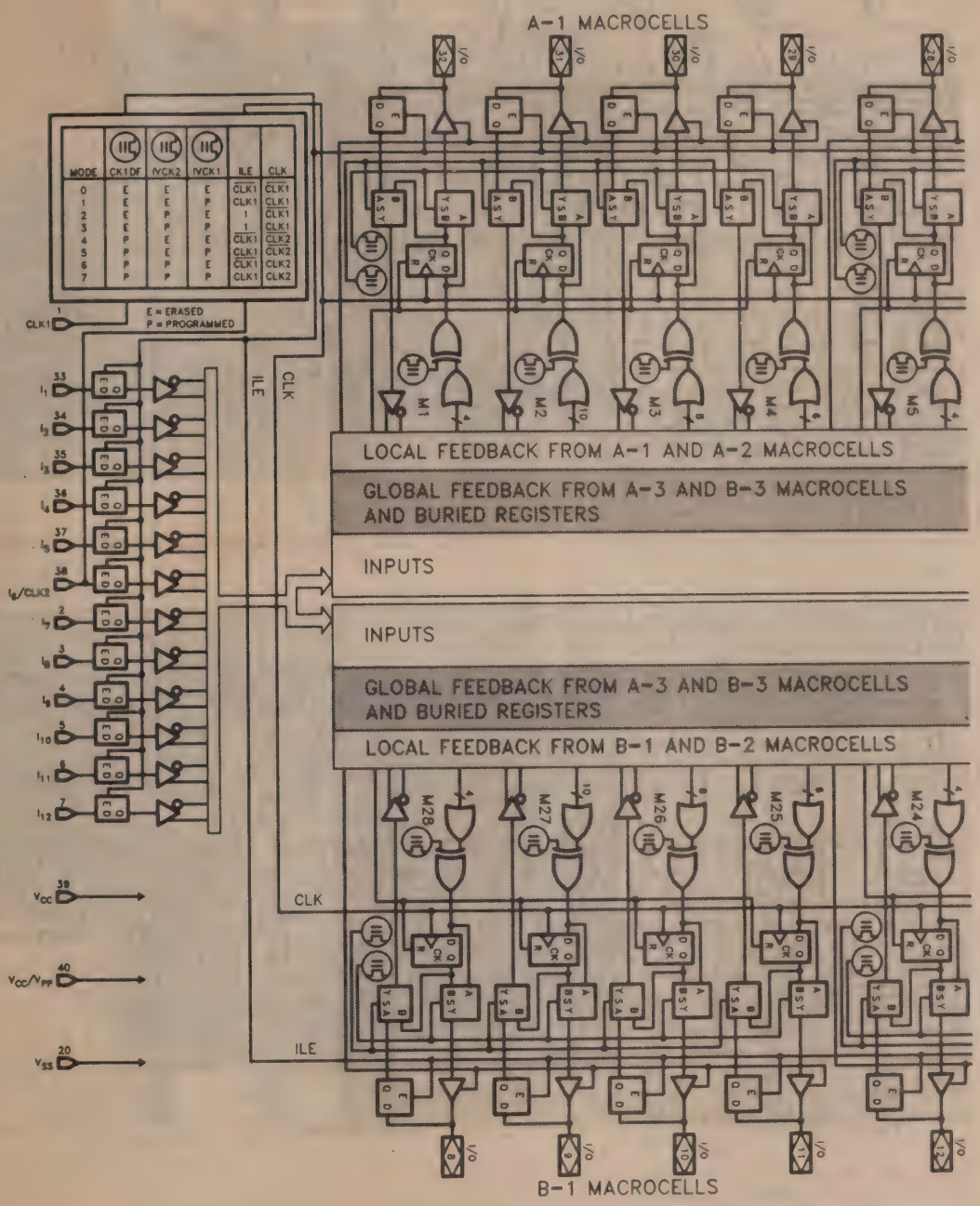


Figure 2. 5C121 Block Diagram

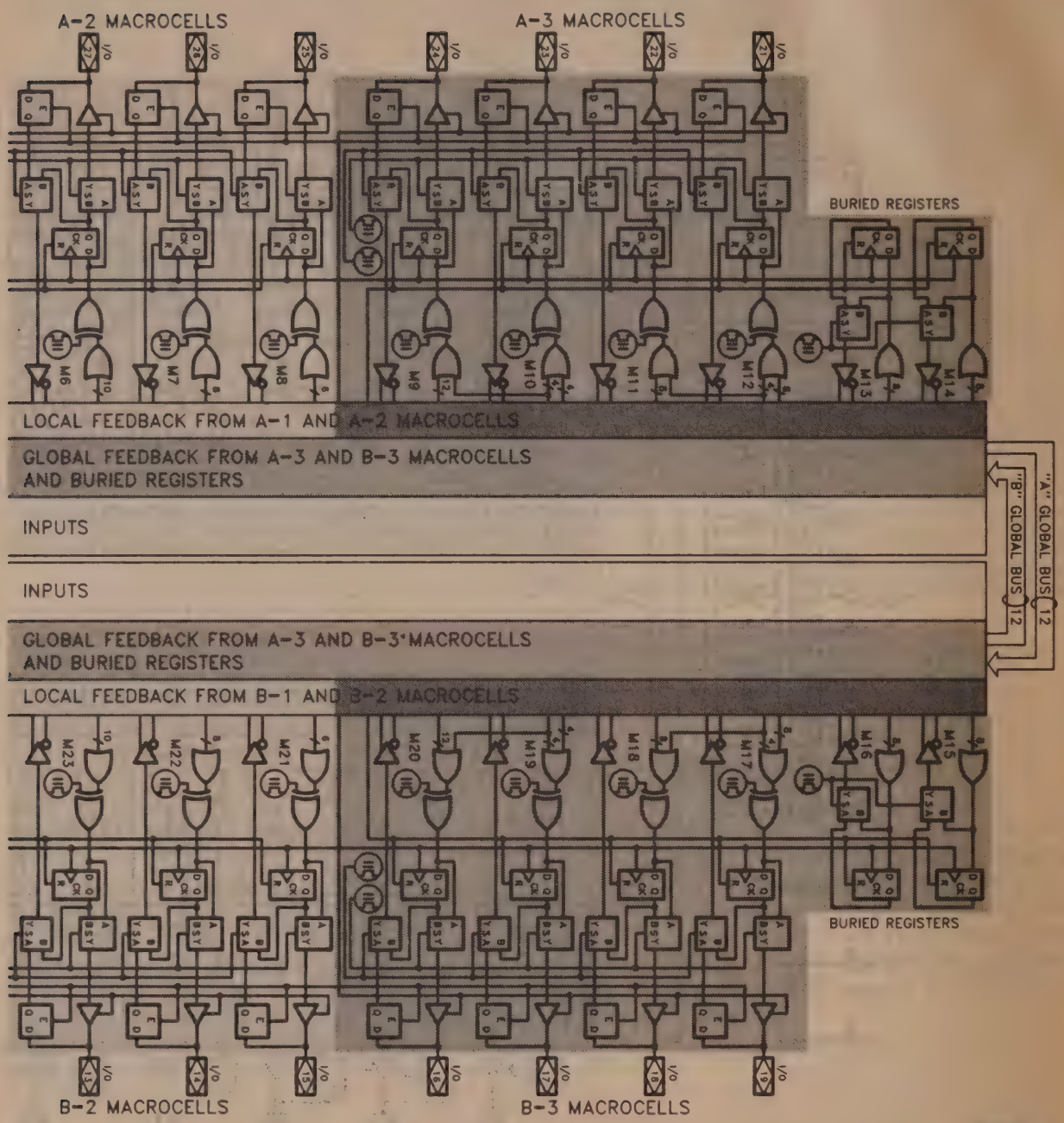


Figure 2. 5C121 Block Diagram (Continued)

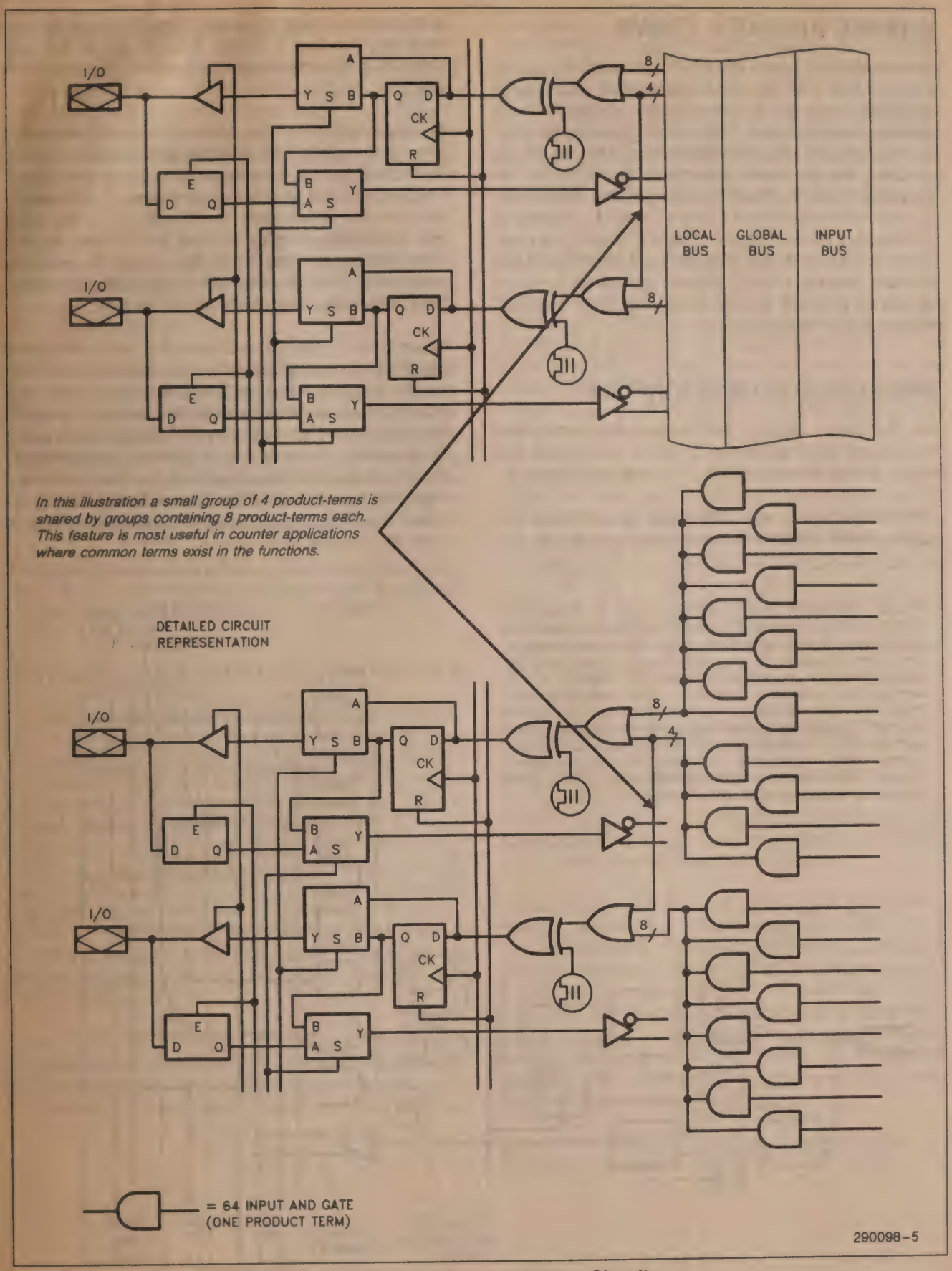


Figure 3. Shared Product-Term Circuits

SHARED PRODUCT TERMS

Macrocells 9 & 10, 11 & 12, 17 & 18 and 19 & 20 (in groups A-3 and B-3—the macrocells with global feedback) have the facility to share a total of 16 additional product terms. This sharing takes place between pairs of adjacent macrocells. This capability enables, for example, macrocells 9 and 10 to expand to 16 and 8 effective product terms respectively, and for macrocells 11 and 12 both to expand to 12 effective product terms. Figure 3 shows this sharing technique in detail. This facility is primarily of use in state machine and counter applications where common product terms are frequently required among output functions.

MACROCELL-BUS INTERFACE

As discussed earlier, the macrocells within the 5C121 are interconnected to other macrocells and inputs to the device via three internal data busses.

The product terms span the entire bus structure (local feedback, global feedback and input buses) that is adjacent to their macrocell (see Figure 4) so that they may produce a logical AND of any of the variables (or their complements) that are present on the busses.

All macrocells have the ability to return data to the local or the global bus. Feedback data may originate from the output of the macrocell or from the I/O pin. Feedback to the global bus communicates throughout the part. Macrocells that feedback to the local bus communicate only to their half of the 5C121. Connections to and from the signal busses are made with EPROM switches that provide the reprogrammable logic capability of the circuit.

Macrocells in groups A-3 and B-3 and the buried registers all have global bus connections while macrocells in groups A-1, A-2 and B-1, B-2 have only local bus connections (see Block Diagram, Figure 2). Advanced features of the Intel Programmable Logic Development System will, if desired, automatically select an appropriate macrocell to meet both the logic requirements and the connection to an appropriate signal bus to achieve the interconnection to other macrocells.

At each intersecting point in the logic array there exists an EPROM-type programmable connection. Initially, all connections are complete. This means that both the true and complement of all inputs are connected to each product-term. Connections are opened during the programming process. Therefore any product term can be connected to the true or complement of any input. When both the true and complement connections of any input are left intact, a logical false results on the output of the AND gate. If both the true and complement connections of any input are programmed open, then a logical "don't care" results for that input. If all inputs for a product term are programmed open, then a logical true results on the output of the AND gate.

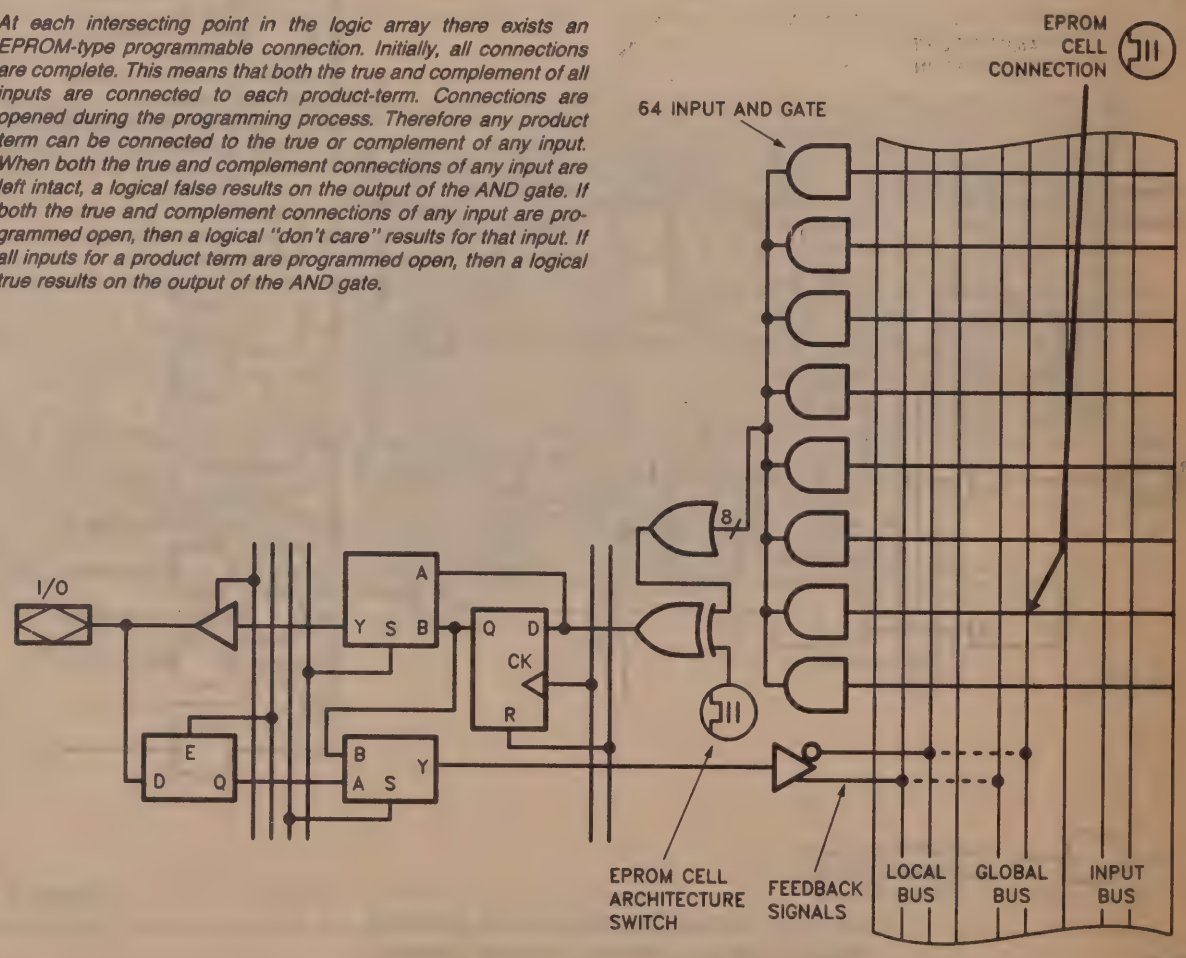


Figure 4. Macrocell-Bus Interface

CLOCK MODE CONTROL

The 5C121 contains two internal clock data paths that drive the input latches (transparent 7475 type) and the output registers. These clocks may be programmed into one of 8 operating modes (see clock mode Table 1). Figure 1 shows a typical macrocell which is driven by the master clock signal CLK and the input latch-enable signal ILE.

The master clock signal is input via pin 1. If programmed modes 4, 5, 6 & 7 are chosen, a second clock signal is required which is input via pin 38 (see Figure 5). Table 1 shows the operation of each clock programming mode.

If modes 0, 1, 4, 5, 6 or 7 are chosen (i.e. latching of the inputs is required), all inputs, both dedicated and I/O, are latched with the same ILE signal. Data applied to the inputs when CLK1 is low (high) is latched when CLK1 goes high (low) and will stay latched as long as CLK1 stays high (low). Levels shown in parenthesis are for modes 1, 5 & 7 and levels shown outside parenthesis are for modes 0, 4 & 6.

Care is required when using any of the clock modes 4, 5, 6 or 7, that require two input clock signals to ensure that timing hazards are not created.

ERASURE CHARACTERISTICS

The erasure characteristics of the 5C121 are such that erasure begins to occur upon exposure to light with wavelengths shorter than approximately 4000Å. It should be noted that sunlight and certain types of fluorescent lamps have wavelengths in the 3000–4000Å. Data shows that constant exposure to room level fluorescent lighting could erase the typical 5C121 in approximately three years, while it would take approximately one week to cause erasure when exposed to direct sunlight. If the 5C121 is to be exposed to these types of lighting conditions for extended periods of time, conductive opaque labels should be placed over the window to prevent unintentional erasure.

The recommended erasure procedure for the 5C121 is exposure to shortwave ultraviolet light which has the wavelength of 2537Å. The integrated dose (i.e., UV intensity × exposure time) for erasure should be a minimum of fifteen (15) Wsec/cm^2. The erasure time with this dosage is approximately 15 to 20 minutes using an ultraviolet lamp with a 12,000 μW/cm^2 power rating. The 5C121 should be placed within one inch of the lamp tubes during erasure. The maximum integrated dose the 5C121 can be exposed to without damage is 7258 Wsec/cm^2 (1 week @ 12,000 μW/cm^2). Exposure to high intensity UV light for longer periods may cause permanent damage.

FUNCTIONAL TESTING

Since the logical operation of the 5C121 is controlled by EPROM elements, the device is completely factory tested. Each programmable EPROM bit controlling the internal logic including the buried state registers are tested using application-independent test program patterns and erased before shipment to customers.

To enable functional evaluation of counter and state-machine applications, the 5C121 contains register pre-load circuitry. This can be activated by interrupting the normal clocked sequence and applying V_{PP} on pin 2 to engage the pre-load state. Under these conditions the flip flops in the 5C121 can be set to any logical condition and then return to normal operation. This process simplifies the input sequences necessary to evaluate the counter and state machine operations.

DESIGN RECOMMENDATIONS

For proper operation it is recommended that input and output pins be constrained to the range GND < (V_{IN} or V_{OUT}) < V_{CC}. Unused inputs should be tied to an appropriate logic level (e.g. either V_{CC} or GND) to minimize device power consumption.

When utilizing a macrocell with an I/O pin connection as a buried macrocell (i.e. just using the macrocell for feedback purposes to other macrocells), its I/O pin is a 'reserved pin'. (The Intel Programmable Logic Development System will label the pin 'RESERVED' in the utilization report that it generates.) Such an I/O pin will actually be an output pin and should not be grounded. It should be left unconnected such that it can go high or low depending on the state of the macrocell's output.

In normal operation V_{CC}/V_{PP} (pin 40) should be connected directly to V_{CC} (pin 39).

Table 1. Clock Programming (Key: L = Latched; T = Transparent)

Programmed Mode	Input Signals Are Latched When:	Output Registers Change State When:	Clock Configuration
0	CLK1 (Pin 1) L T	CLK1 (Pin 1)	1 Clock
1	CLK1 (Pin 1) T L	CLK1 (Pin 1)	1 Clock
2	Inputs Not Latched	CLK1 (Pin 1)	1 Clock
3	Inputs Not Latched	CLK1 (Pin 1)	1 Clock
4	CLK1 (Pin 1) L T	CLK2 (Pin 38)	2 Clocks
5	CLK1 (Pin 1) T L	CLK2 (Pin 38)	2 Clock
6	CLK1 (Pin 1) L T	CLK2 (Pin 38)	2 Clocks
7	CLK1 (Pin 1) T L	CLK2 (Pin 38)	2 Clocks

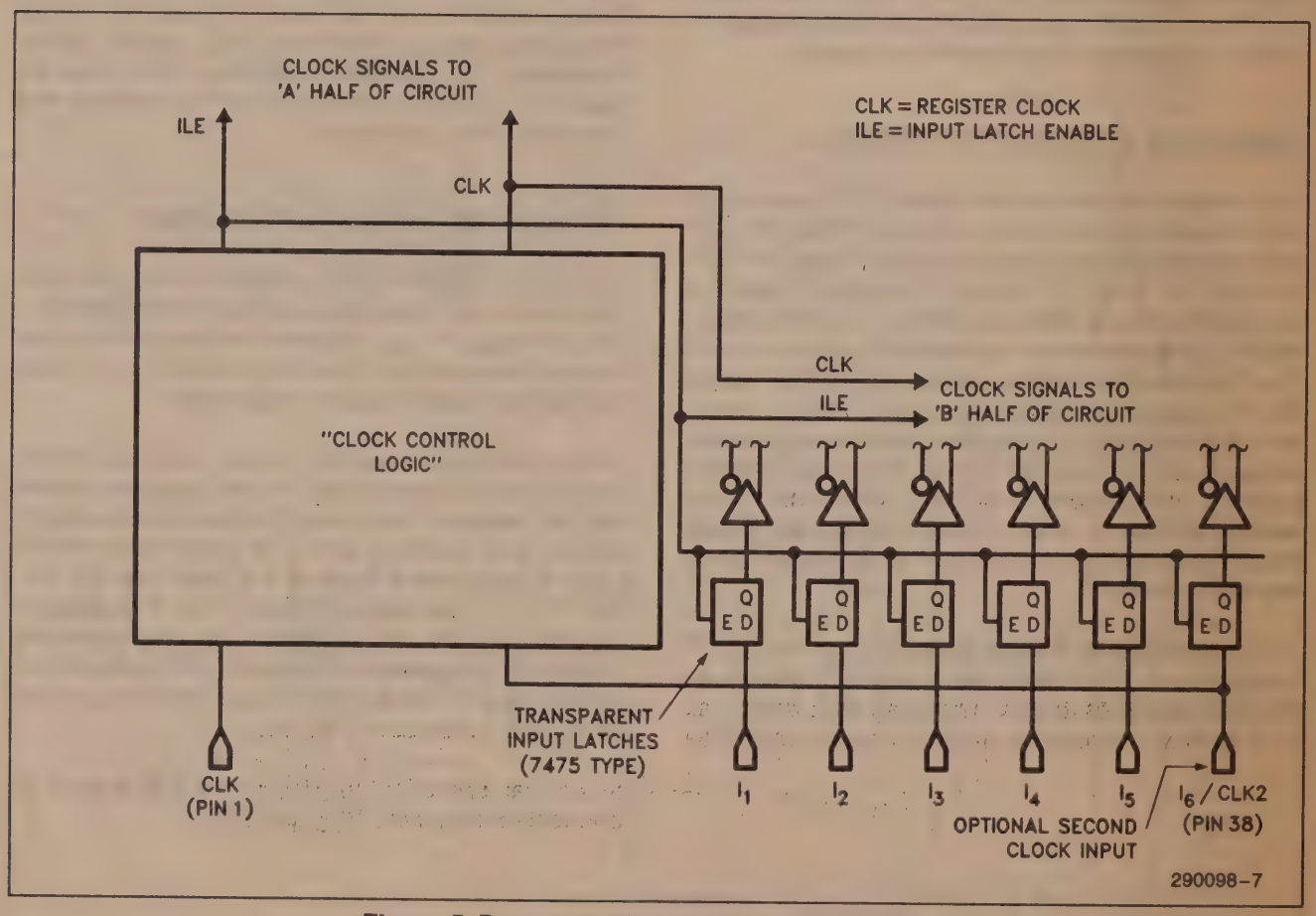


Figure 5. Programmable Clock Control System

ABSOLUTE MAXIMUM RATINGS*

Symbol	Parameter	Min	Max	Units
V_{CC}	Supply Voltage(1)	−2.0	7.0	V
V_{PP}	Programming Supply Voltage(1)	−2.0	13.5	V
V_I	DC Input Voltage(1)(2)	−0.5	V_{CC}+0.5	V
I_{CC}	DC V_{CC} Current(4)		100	mA
T_{stg}	Storage Temperature	−65	+150	°C
T_{amb}	Ambient Temperature(3)	−10	+85	°C

NOTES:
1. Voltages with respect to ground.
2. Minimum DC input is −0.3V. During transitions, the inputs may undershoot to −0.2V for periods less than 20 ns under no load conditions.
3. Under bias.
4. With outputs tristated.

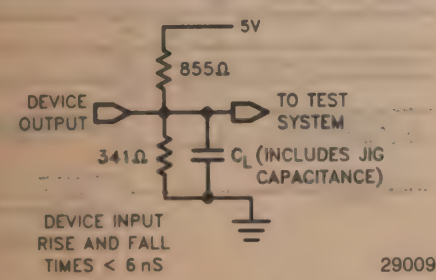
**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS T_A = 0° to 70°C, V_{CC} = 5.0V ±5%

Symbol	Parameter	Conditions		Min	Typ	Max	Unit
V_{IH}	HIGH Level Input Voltage			2.0		V_{CC}+0.3	V
V_{IL}	LOW Level Input Voltage			−0.3		0.8	V
V_{OH}	HIGH Level Output Voltage	I_O = −4.0 mA DC		2.4			V
V_{OL}	LOW Level Output Voltage	I_O = 4.0 mA DC				0.45	V
I_I	Input Leakage Current	V_I = V_{CC} or GND				±10.0	μA
I_{OZ}	3-State Output Off-State Current	V_O = V_{CC} or GND				±10.0	μA
I_{CC1}	V_{CC} Supply Current (Standby) (Note 6)	V_I = V_{CC} or GND, I_O = 0	CMOS Inputs			3	mA
			TTL Inputs			30	
I_{CC2}	V_{CC} Supply Current (Active)	No Load, f = 10 MHz	CMOS Inputs			50	mA
			TTL Inputs			100	
I_{OS}	Output Short Circuit Current	(Note 5)				130	mA

NOTE:
5. Output shorted for no more than 1 sec. and no more than one output shorted at a time. I_{OS} is sampled but not 100% tested.
6. Chip automatically goes into standby mode if logic transitions do not occur. (Approximately 50 ns after last transition.)

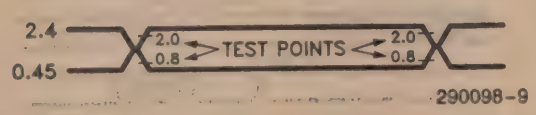
A.C. TESTING LOAD CIRCUIT



290098-8

C_L = 30 pF
C_L Includes Jig Capacitance

A.C. TESTING INPUT, OUTPUT WAVEFORM



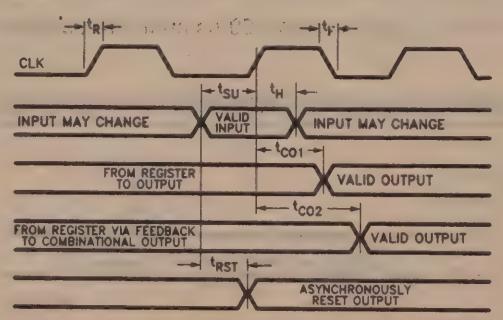
290098-9

A.C. Testing: Inputs are Driven at 2.4V for a Logic "1" and 0.45V for a Logic "0". Timing Measurements are made at 2.0V for a Logic "1" and 0.8V for a Logic "0".

A.C. CHARACTERISTICS T_A = 0° to 70°C, V_{CC} = 5.0V ±5%

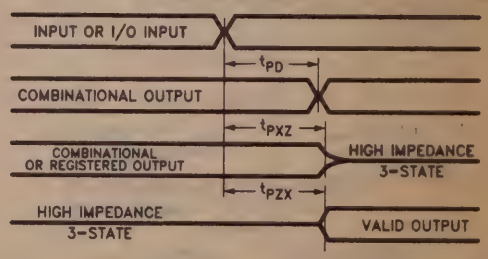
Symbol	Parameter	Device Conditions	5C121-50 Min	5C121-50 Max	5C121-65 Min	5C121-65 Max	5C121-90 Min	5C121-90 Max	Unit
t_{PD}	Non-Registered Input or I/O Input to Non-Registered Output			50		65		90	ns
t_{PZX}	Non-Registered Input or I/O Input to Output Enable	C_L = 30 pF		50		65		90	ns
t_{PXZ}	Non-Registered Input or I/O Input to Output Disable			50		65		90	ns
t_{SU}	Non-Registered Input or I/O Input to Output Register Setup		37		47		62		ns
t_H	Non-Registered Input or I/O Input to Output Register Hold		0		0		0		ns
t_{CH}	Clock High Time		20		25		30		ns
t_{CL}	Clock Low Time	C_L = 30 pF	20		25		30		ns
t_{CO1}	Clock to Output Delay			28		33		38	ns
t_{P1}	Minimum Clock Period (Register Output Feedback to Register Input—Internal Path)			50		55		75	ns
f_1	Maximum Frequency ($1/t_{P1}$)		20.0		18.2		13.3		MHz
t_{P2}	Minimum Clock Period ($t_{SU} + t_{CO1}$)			65		80		100	ns
f_2	Maximum Frequency ($1/t_{P2}$)		15.0		12.5		10.0		MHz
t_{RST}	Asynchronous Reset Time			50		65		90	ns
t_{CO2}	Registered Feedback Through PLA to output. Relative to External Clock.			70		75		100	ns
t_{ILS}	Set Up Time for Latching Inputs		0		0		0		ns
t_{ILH}	Hold Time for Latching Inputs		15		20		25		ns
t_{C1C2}	Minimum Clock 1 to Clock 2 Delay			40		50		65	ns
t_{ILDFS}	Input Latch to D-FF Setup Time	Mode 0, 1	40		50		65		ns
t_{DFILS}	D-FF to Input Latch Setup Time	Mode 0, 1	25		30		35		ns
t_{P3}	Minimum Period for a 2-Clock System ($T_{C1C2} + t_{CO1}$)			65		85		100	ns
f_3	Maximum Frequency ($1/t_{P3}$)		15.0		12.0		10.0		MHz

SWITCHING WAVEFORMS



290098-10

NOTES:
t_R & t_F = 6 ns
t_{CL} & t_{CH} measured at 0.3V & 2.7V.
All other timing at 1.3V.

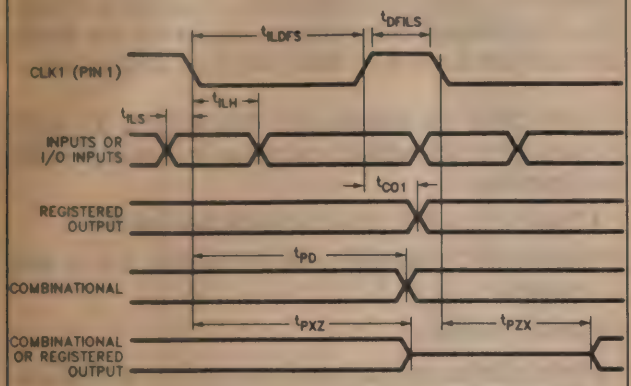


290098-11

NOTE:
Above waveforms shown for clock modes 2 or 3 (t_{SU} & t_H are as in modes 2 & 3; no ILE signal is used).

CLOCK MODES SWITCHING WAVEFORMS

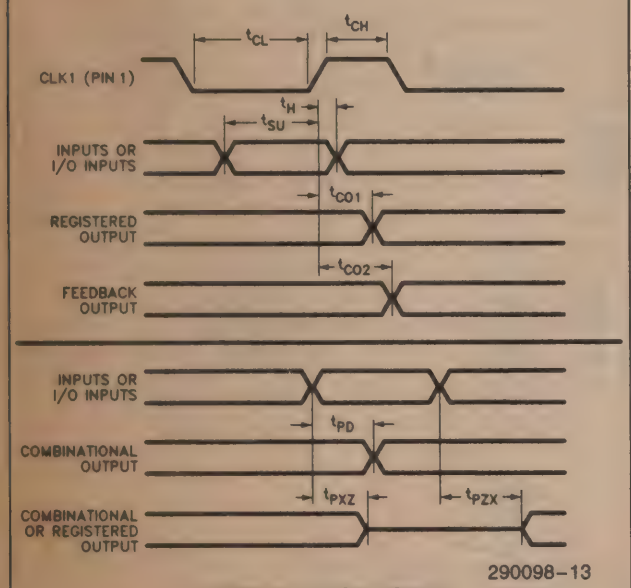
1-CLOCK SYSTEM: MODES 0 AND 1



290098–12

INVERT CLK1 FOR MODE 0

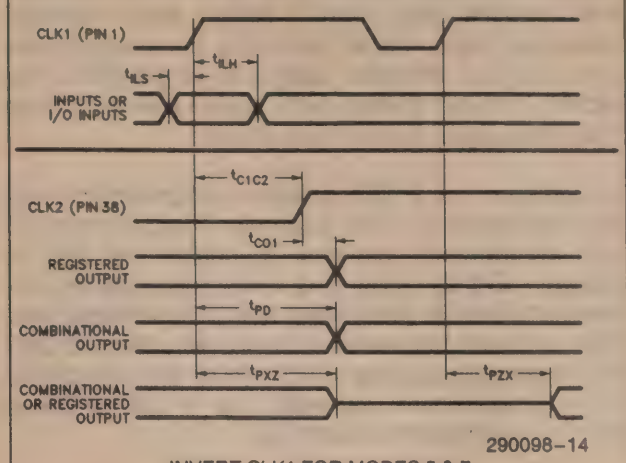
1-CLOCK SYSTEM: MODES 2 AND 3



290098–13

INVERT CLK1 FOR MODE 2

2-CLOCK SYSTEMS: MODES 4 THROUGH 7
CLK1 PIN 1 CONTROLS THE INPUT LATCH CLOCK
CLK2 PIN 38 CONTROLS THE D-FF CLOCK.



290098–14

INVERT CLK1 FOR MODES 5 & 7
INVERT CLK2 FOR MODES 4 & 5

Intel Programmable Logic Development System (iPLDS)

The iPLDS provides all the tools needed to design with Intel H-Series EPLDs or compatible devices. It contains comprehensive third generation software that supports four different design entry methods, minimizes logic, does automatic pin assignments and produces the best design fit for the EPLD selected. It is user friendly with guided menus, on-line Help messages and soft key inputs.

In addition, the iPLDS contains programmer hardware in the form of an expansion card for the PC with programming software to enable the user to program EPLDs, read and verify programmed devices and also to graphically edit programming files. The software generates industry standard JEDEC object code output files which can be downloaded to other programmers as well. The iPLDS includes 5C121 H-series EPLD samples.

The iPLDS has interfaces to popular schematic capture packages (Dash series from Futurenet and PC CAPS from PCAD) to enable designs to be entered using schematics. However, hand-drawn schematics can be entered just as easily using the Logic Builder, interactive netlist entry program included in iPLDS. The other design entry formats supported are Boolean equation entry and State Machine design entry.

The iPLDS is compatible with IBM PC, PC XT or PC AT and other equivalent machines with the following configuration:

(1) Dual floppy disk drive or hard disk drive

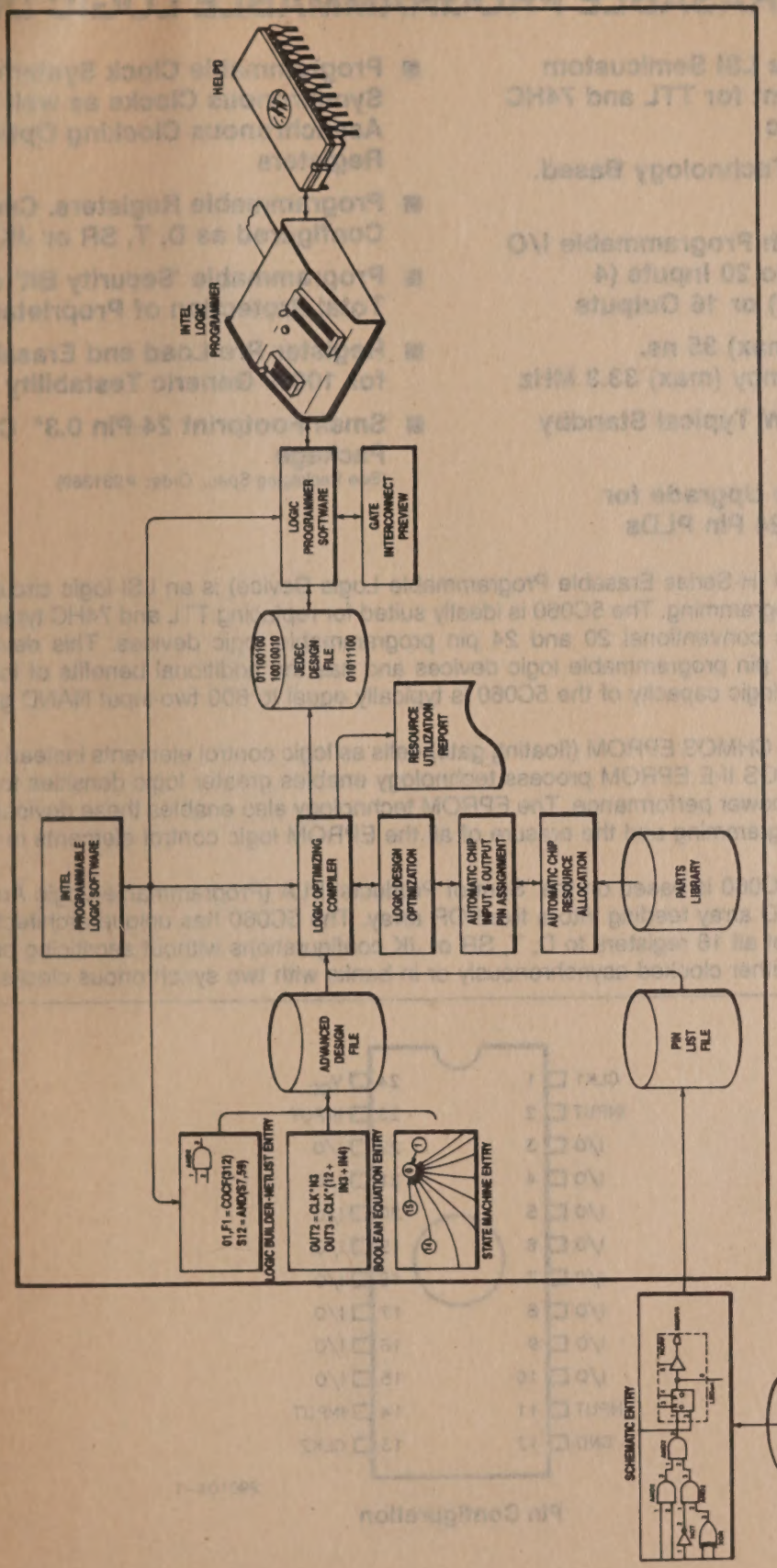
(2) MS-DOS Operating System Version 2.0 or later release

(3) 384K Memory

(4) Intel device programming card and unit (supplied with iPLDS).

Detailed information on the Intel Programmable Logic Development System is contained in a separate Intel data sheet on this product.

IPLDS INTEL PROGRAMMABLE LOGIC DEVELOPMENT SYSTEM



290098-19

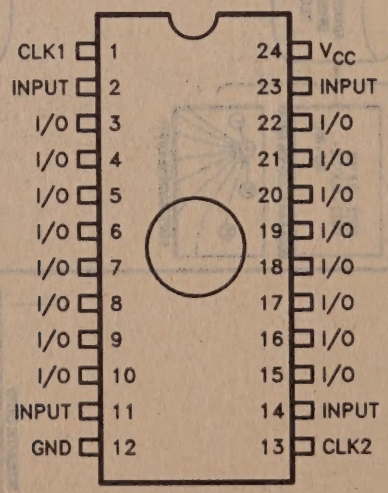
5C060
600 GATE CHMOS
H-SERIES ERASABLE PROGRAMMABLE LOGIC DEVICE

- **High Performance LSI Semicustom Logic Replacement for TTL and 74HC SSI and MSI Logic**
- **CHMOS EPROM Technology Based. UV Erasable**
- **16 Macrocells with Programmable I/O Architecture; up to 20 Inputs (4 Dedicated, 16 I/O) or 16 Outputs**
- **High Speed t_{pd} (max) 35 ns. Operating Frequency (max) 33.3 MHz**
- **Low Power; 10 μW Typical Standby Dissipation**
- **High Performance Upgrade for Commonly Used 24 Pin PLDs**
- **Programmable Clock System with Two Synchronous Clocks as well as Asynchronous Clocking Option on All Registers**
- **Programmable Registers. Can be Configured as D, T, SR or JK Types**
- **Programmable 'Security Bit' Allows Total Protection of Proprietary Designs**
- **Register Pre-Load and Erasable Array for 100% Generic Testability**
- **Small Footprint 24-Pin 0.3" Cerdip Package**
  (See Packaging Spec., Order #231369)

The Intel 5C060 H-EPLD (H-Series Erasable Programmable Logic Device) is an LSI logic circuit that is user customizable through programming. The 5C060 is ideally suited for replacing TTL and 74HC type SSI and MSI logic devices as well as conventional 20 and 24 pin programmable logic devices. This device is socket compatible with most 24 pin programmable logic devices and has the additional benefits of low power and increased flexibility. The logic capacity of the 5C060 is typically equal to 600 two-input NAND gates.

The 5C060 H-ELPD uses CHMOS EPROM (floating gate) cells as logic control elements instead of fuses. Use of Intel's advanced CHMOS II-E EPROM process technology enables greater logic densities to be achieved with superior speed and power performance. The EPROM technology also enables these devices to be 100% factory tested by the programming and the erasure of all the EPROM logic control elements in the device.

The architecture of the 5C060 is based on the 'Sum of Products' PLA (Programmable Logic Array) structure with a programmable AND array feeding into a fixed OR array. The 5C060 has unique architectural features that allow programming of all 16 registers to D, T, SR or JK configurations without sacrificing product terms. These registers can be either clocked asynchronously or in banks with two synchronous clocks.



290104-1

Pin Configuration

Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied. Information contained herein supersedes previously published specifications on these devices from Intel.
© Intel Corporation, 1985

November 1985
Order Number: 290104-001

intel

UNITED STATES
Intel Corporation
3065 Bowers Avenue
Santa Clara, CA 95051

JAPAN
Intel Japan K.K.
5-6 Tokodai Toyosato-machi
Tsukuba-gun, Ibaraki-ken 300-26
Japan

FRANCE
Intel Paris
1 Rue Edison, BP 303
78054 Saint-Quentin en Yvelines
France

UNITED KINGDOM
Intel Corporation (U.K.) Ltd.
Piper's Way
Swindon
Wiltshire, England SN3 1RJ

WEST GERMANY
Intel Semiconductor GmbH
Seidlstrasse 27
D-8000 Munchen 2
West Germany

Printed in U.S.A. /C421/1285/100K/RRD LD PM
Microprocessors/Peripherals

ISBN 1-555-12-001-6